Relaxation de contraintes globales : Mise en oeuvre et Application

Jean-Philippe Metivier

UNIVERSITÉ de CAEN/BASSE-NORMANDIE

U.F.R. : Sciences

ÉCOLE DOCTORALE : SIMEM

THÈSE

présentée par

Jean-Philippe MÉTIVIER

et soutenue

le Vendredi 9 avril 2010

en vue de l'obtention du

DOCTORAT de l'UNIVERSITÉ de CAEN

spécialité : Informatique et applications

(Arrêté du 7 août 2006)

Relaxation de contraintes globales : Mise en œuvre et Application

MEMBRES du JURY

Christian BESSIÈRE	Directeur de recherche	LIRMM	(rapporteur)
Narendra JUSSIEN	Professeur	École de Mines de Nantes	(rapporteur)
Simon DE GIVRY	Chargé de recherche	INRA Toulouse	
Etienne GRANDJEAN	Professeur des universités	Université de Caen Basse-Normandie	
Arnaud LALLOUET	Professeur des universités	Université de Caen Basse-Normandie	
Thierry PETIT	Maître de conférences	École de Mines de Nantes	
Patrice BOIZUMAULT	Professeur des universités	Université de Caen Basse-Normandie	(directeur)
Samir LOUDNI	Maître de conférences	Université de Caen Basse-Normandie	(co-encadrant)

Mis en page avec la classe thloria.

Table des matières

À tous mes amis,

Je tiens à remercier mes deux directeurs de thèse, Patrice Boizumault et Samir Loudni, sans qui cette thèse n'aurai pas eu lieu et n'aurai certainement pas abouti au présent mémoire. Je tiens surtout à les remercier pour leur grande patience notamment lors de la phase finale de cette thèse.

Je remercie aussi tous les membres du jury pour l'intérêt qu'ils ont porté à mes travaux de thèse, avec un remerciement tout particulier pour Thierry Petit pour les nombreuses discussions que nous avons eu en début de thèse qui m'ont été d'une grande aide.

Je tiens aussi à remercier tous les membres présents et passés du bureau 384 que j'ai du faire souffrir avec mes « grignottages de touillettes en plastique ». Je remercie tout d'abord Nicolas Levasseur et Bruno Zanuttini pour les discussions souvent fructueuses que nous avons eu ensemble et aussi pour la bonne ambiance qui a toujours règné dans notre bureau. Je tiens aussi à remercier Jean Fromentin qui a « traversé la frontière » et nous a rejoint durant ma dernière année de thèse.

Je tiens à remercier mes amis de longue date du groupe BTH pour nos étés (video)ludiques qui ont été de véritables oasis durant cette thèse. Mais aussi mes amis de l'IUT de génie chimique qui ont toujours été présents malgré l'éloignement. Je souhaite aussi remercier tous les doctorants et docteurs membres de la liste Fuche pour les nombreuses parties de « Race », « Perudo » et autres qui ont vraiment permi à nos pauses déjeuners d'être aussi rayonnantes (je remercie aussi ceux et celles qui ne jouaient pas).

Je souhaite aussi remercier tout le personnel du GREYC et du département informatique pour leur disponibilité et leur gentillesse.

Finalement, je remercie tous les membres de ma famille ma mère mon père et mes deux soeurs Pauline et Justine que je n'ai pas réussi à voir autant de fois que je l'aurai voulu durant cette thèse.

Introduction

1 Contexte

1.1 Contraintes globales

Une contrainte globale représente généralement un ensemble de propriétés que doit satisfaire un ensemble de variables. Il est souvent très agréable de formuler cette propriété par une seule contrainte plutôt que par une conjonction de contraintes (élémentaires ou pas) qui soit équivalente. Ainsi, la contrainte globale `AllDifferent` impose qu'un ensemble de variables prenne des valeurs deux à deux différentes [Rég94]. De très nombreuses contraintes globales ont été développées depuis une vingtaine d'années (pour un catalogue de ces contraintes, cf. [BCDP07, BD09]). L'apport des contraintes globales est double : elles permettent une modélisation concise et élégante des problèmes et, grâce à l'efficacité de leur filtrage, elles sont souvent des éléments clef dans la résolution de problèmes réels de grande taille.

1.2 Relaxation de problèmes sur-contraints

De nombreux problèmes réels sont par nature sur-contraints : ils ne possèdent aucune solution. Cela est souvent dû à un ensemble d'exigences trop fortes des spécifieurs ou des utilisateurs de l'application, ce qui conduit à l'introduction de contraintes antagonistes. Lorsque l'on cherche à modéliser un problème sur-contraint, on distingue deux catégories de contraintes :

1. les contraintes d'intégrité (ou dures) qui doivent être impérativement satisfaites,
2. les contraintes de préférence (ou molles) qui expriment des souhaits formulés sous forme de propriétés que l'on aimerait voir vérifiées par une solution. Ces contraintes de préférence correspondent à des exigences qu'il est souhaitable de respecter pour obtenir une solution de bonne qualité.

Le but de la relaxation n'est pas de satisfaire toutes les contraintes mais de les satisfaire au mieux. « Au mieux » signifie ici satisfaire toutes les contraintes dures et minimiser une agrégation des coûts des contraintes de préférence insatisfaites. Différents cadres pour la relaxation des Problèmes de Satisfaction de Contraintes (CSP) ont été proposés : les CSPs hiérarchiques [BFBW92], les Partial CSPs [FW92], les CSPs valués [SFV95], les Semi-ring CSPs [BMR+99], la relaxation disjonctive de CSPs [Pet02]. L'ensemble des travaux synthétisés dans

[Jus03] montrent notamment l'apport des explications pour la résolution de problèmes sur-contraints.

Un cadre approprié pour la relaxation [Pet02] doit permettre d'exprimer différents niveaux de priorités (les contraintes de préférence n'ont pas toute la même importance et on essaiera donc de satisfaire les plus importantes), de quantifier la violation de manière fine (niveau d'insatisfaction, nature de la violation) et de contrôler la violation.

1.3 Sémantiques de violation

À la différence d'une contrainte élémentaire, une contrainte globale peut être *plus ou moins violée*. Il faut donc proposer des sémantiques de violation permettant de définir et de quantifier la violation d'une instanciation. Il existe généralement plusieurs sémantiques de violation pour une même contrainte globale [Pet02], comme par exemple, la sémantique basée décomposition (nombre de contraintes élémentaires insatisfaites par la décomposition[1] de la contrainte globale) et la sémantique basée variables (nombre de variables qu'il est nécessaire de ré-instancier pour satisfaire la contrainte).

1.4 Relaxation de contraintes globales dans le cadre disjonctif

Pour une contrainte globale et une sémantique de violation associée, test de cohérence et algorithme de filtrage doivent à la fois utiliser la structure de cette contrainte et la structure de sa violation. En effet, pour les problèmes réels, les violations des contraintes obéissent souvent à certaines règles et ont une certaine structure.

Le cadre disjonctif de Thierry Petit permet de transformer un problème de relaxation, qui est un problème d'optimisation (COP), en un CSP. Les contraintes globales relaxées deviennent des contraintes dures ayant chacune une variable de coût. Une contrainte relaxée est satisfaite si, et seulement si, son coût de violation (selon la sémantique de violation choisie) est compris entre les bornes du domaine de sa variable de coût. Une variable globale de coût est définie comme la somme des variables de coût de chacune des contraintes globales relaxées. C'est ce cadre que nous avons retenu pour nos travaux.

1.5 Travaux sur la relaxation de contraintes globales

Depuis le début des années 2000, différentes relaxations ont été proposées pour les contraintes globales `AllDifferent` [Rég94], `Gcc` [Rég96], `Regular` [Pes04], `Same` [Bel00], `cumulative` [BC94] et `Grammar` [QW06]. Même si initialement ces travaux n'ont pas tous été présentés *stricto sensu* dans le cadre de la relaxation disjonctive, ils peuvent être aisément et naturellement formulés dans ce cadre.

Willem-Jan van Hoeve *et al.* [Hoe04, Hoe05, HPR06] ont proposé différentes relaxations des contraintes `AllDifferent` et `Gcc` pour la sémantique basée variables et la sémantique basée

[1] Si elle existe !

décomposition, ainsi que différentes relaxations de la contrainte `Regular` pour des sémantiques spécifiques basées sur les distances entre mots. Pour mettre en œuvre test de cohérence et filtrage, les auteurs réutilisent la représentation sous forme de réseau de ces contraintes en ajoutant des arcs, dit de violation. Les arcs de violation permettent de modéliser et quantifier la violation pour une sémantique donnée. Mais, cette approche n'est pas la seule utilisable. Dans certains cas, il est possible d'exploiter les mêmes algorithmes dans le cadre disjonctif [Pet02].

2 Contributions

2.1 Motivations

Les sémantiques de violation proposées jusque là manquent d'expressivité pour modéliser de véritables problèmes réels. En effet, elles considèrent que toutes les violations sont de même niveau. Ainsi, pour la sémantique basée variables, toutes les variables à ré-instancier sont considérées de même importance (seul leur nombre importe). De même, pour la sémantique basée décomposition, toutes les contraintes élémentaires appartenant à la décomposition d'une contrainte globale sont considérées de même importance (là encore, seul leur nombre importe). Il en est de même pour les sémantiques de violation spécifiques à `Regular`.

Exemple : Considérons les deux règles suivantes (issues d'un problème d'emplois du temps des infirmières dans les hôpitaux) telles qu'elles sont spécifiées dans [VH00] :

1. Sur une période de 28 jours, une infirmière doit avoir entre 10 et 13 jours de repos. Tout jour de repos manquant entraîne un coût de 1000 et tout jour de repos excédentaire un coût de 100.
2. Les séquences de travail d'une infirmière doivent être de longueur de 4. Tout jour de travail excédentaire entraîne un coût de 1000. Les séquences de travail composées d'une journée sont interdites et celles de longueur 2 ou 3 sont respectivement pénalisées d'un coût de 20 et de 40.

Si la première règle peut se modéliser par une version relaxée de `Gcc`, mesurer l'écart aux bornes ne suffit pas. Il est nécessaire de différencier les coûts de violation des bornes supérieures et des bornes inférieures. Si pour la seconde règle un automate permet de modéliser les séquences autorisées, les violations devront être traduites par l'ajout de nouvelles transitions (associées aux séquences non autorisées) ayant chacune un coût approprié.

L'objectif de ce travail est d'étudier la relaxation de contraintes globales dans un cadre où il est possible d'exprimer des préférences. En effet, toutes les contraintes n'ont pas la même importance et il est nécessaire de quantifier la violation de manière fine selon le niveau d'insatisfaction et la nature de la violation. Ces préférences sont modélisées grâce aux sémantiques de violation, en associant pour `AllDifferent` et `Gcc`, des poids aux variables et aux contraintes élémentaires issues d'une décomposition, et par des poids sur des transitions non autorisées pour `Regular`.

Enfin, si de nombreux travaux portent sur les technologies de mise en œuvre de la relaxation de contraintes globales (cf. Section 1.5), il n'existe pas, à notre connaissance, d'application réelle de grande taille reposant sur une telle relaxation.

2.2 Contributions

Dans ce mémoire, nous étudions tout d'abord la relaxation des contraintes `AllDifferent`, `Gcc` et `Regular` dans un cadre avec préférences. Nous définissons plusieurs sémantiques de violation et nous montrons comment mettre en œuvre test de cohérence et filtrage associés. Parmi le grand nombre de contraintes globales existantes, nous avons choisi d'étudier plus particulièrement ces trois contraintes car elles figurent parmi les contraintes globales les plus connues et les plus utilisées dans le développement d'applications. Il s'agit aussi des contraintes sur lesquelles ont porté les principaux travaux sur la relaxation des contraintes globales.

Dans un second temps, nous montrons l'apport des contraintes globales relaxées pour la modélisation et la résolution des problèmes d'allocation d'emplois du temps des infirmières dans les hôpitaux (« Nurse Rostering Problems », `NRPs`). Notre approche permet une modélisation élégante et concise des `NRPs`, et malgré sa généricité, elle offre des temps de résolution proches de ceux obtenus par des méthodes *ad'hoc* de Recherche Opérationnelle. Ce travail nous a amené à étudier la communication entre les filtrages des contraintes globales relaxées. Pour améliorer l'efficacité du filtrage, nous proposons une nouvelle contrainte globale permettant d'agglomérer une contrainte `Regular` avec une contrainte de cardinalité (`atmost` et/ou `atleast`).

2.3 Annonce du plan

Le mémoire est classiquement organisé en deux parties : la première consacrée à l'état de l'art et la seconde dévolue aux contributions.

Etat de l'art : Le chapitre 1 rappelle les notions de base relatives aux CSPs et à la théorie des graphes dont nous aurons besoin dans ce mémoire. Le chapitre 2 décrit la sémantique et la mise en œuvre des trois contraintes globales `AllDifferent`, `Gcc` et `Regular` que nous avons choisi d'étudier. Le chapitre 3 présente différents cadres pour la relaxation de contraintes et motive notre choix du cadre de la relaxation disjonctive [Pet02] pour la relaxation des contraintes globales. Le chapitre 4 dresse un état de l'art de la relaxation des trois contraintes globales `AllDifferent`, `Gcc` et `Regular`.

Contributions : Dans le **chapitre 5**, nous étudions deux sémantiques de relaxation pour la contrainte `AllDifferent` dans le cadre avec préférences. Pour la première sémantique (basée variables), le coût de violation d'une instanciation complète est la somme des poids des variables à ré-instancier pour satisfaire la contrainte. Pour cette sémantique, test de cohérence globale et filtrage peuvent être mis en œuvre en temps polynomial grâce aux arcs de violation. Pour la seconde sémantique (basée décomposition), le coût de violation d'une instanciation complète est

la somme des poids des contraintes binaires de différence insatisfaites. Pour cette sémantique, nous montrons que le test de cohérence globale est un problème NP-Complet. C'est pourquoi, nous proposons de maintenir une cohérence plus faible en temps polynomial grâce au calcul d'un minorant, dont nous évaluerons expérimentalement la qualité.

Dans le **chapitre 6**, nous présentons les relaxations des contraintes `Gcc` et `Regular` dans le cadre avec préférences. Pour la contrainte `Gcc`, nous étudions deux sémantiques de violation : la première qui mesure la somme des poids des variables à ré-instancier et la seconde qui mesure la somme pondérée des écarts aux bornes. Pour la contrainte `Regular`, nous étudions également deux sémantiques de violation : la première basée sur une version pondérée de la distance de Hamming et la seconde basée sur une version pondérée de la distance d'édition. Pour chacune de ces quatre sémantiques, test de consistance et filtrage associés sont de complexité polynomiale et peuvent être mis en œuvre en ajoutant aux réseaux et graphes en couches sous-jacents des arcs de violation. Nous concluons ce chapitre en montrant les liens entre ces versions relaxées et les versions « optimisation » de ces deux contraintes que sont `costGcc` [Rég02] et `costRegular` [DPR06].

Le **chapitre 7** traite du manque de communication entre les filtrages des contraintes globales relaxées. Nous présentons les deux principales approches permettent d'améliorer cette communication : le transport de coûts entre les contraintes du réseau et la définition de contraintes globales agglomérantes. Puis, nous présentons nos travaux permettant d'agglomérer une contrainte `Regular` avec une contrainte de cardinalité (`atmost` et/ou `atleast`). Nous montrerons l'intérêt et les apports d'une telle contrainte (et de sa version relaxée) sur un ensemble de problèmes d'affectation de personnels.

Le **chapitre 8** est consacré à modélisation et à la résolution des `NRPs` dont la très grande majorité sont des problèmes sur-contraints. Les `NRPs` ont été intensivement étudiés depuis une quarantaine d'années aussi bien par la communauté Recherche Opérationnelle que par la communauté Intelligence Artificielle. La plupart des `NRPs` sont des problèmes NP-complets et sont aussi particulièrement difficiles à résoudre en raison de la grande diversité des règles et des préférences que doit respecter un planning. Résoudre un `NRP` consiste à générer un planning associant un ensemble d'équipes (Matin, Midi, Soir, ...) aux infirmières d'un service sur une période donnée, tout en respectant un ensemble de contraintes d'intégrité et de contraintes de préférence. Ces contraintes sont généralement issues de la législation du travail, des besoins des hôpitaux ou encore des préférences émises par le personnel hospitalier.

Dans ce chapitre, nous motivons et étudions l'apport des contraintes globales relaxées pour la modélisation et la résolution des `NRPs`. Nous montrons que notre approche permet une modélisation élégante et concise des `NRPs`, et que malgré sa généricité, elle offre des temps de résolution proches de ceux obtenus par des méthodes *ad'hoc* sur de nombreuses instances issues du site du ASAP (Automated Scheduling, Optimisation and Planning) de l'université de Nottingham qui sert de référence à la communauté Recherche Opérationnelle.

Dans le **dernier chapitre**, nous concluons et dressons différentes perspectives prolongeant ce travail.

Première partie

État de l'art

Chapitre 1

CSP et graphe : notions de base

Sommaire

Ce chapitre introductif présente les notions de base relatives aux CSPs et à la théorie des graphes, dont nous aurons besoin dans ce mémoire.

Tout d'abord, nous présentons les notions de CSP, de cohérence et les mécanismes de filtrage associés, ainsi que différentes méthodes de recherche. Puis, nous rappelons, de manière très synthétique, les notions de graphes (composantes connexes, couplages) et de réseaux (flots). Enfin, nous présentons brièvement la notion d'automate.

1.1 Le formalisme des CSPs

1.1.1 Variables et domaines

Soit $\mathcal{X}$ un ensemble fini de n variables $\mathcal{X}=\{X_1, ..., X_n\}$. À chaque variable X_i est associée un domaine, noté D_{X_i}, représentant l'ensemble fini des valeurs pouvant être prises par cette variable. Le domaine d'une variable peut être numérique {1,3,4} ou symbolique {Matin,Soir,Repos}. On désigne l'ensemble des domaines par $\mathcal{D} = \{D_{X_1}, ..., D_{X_n}\}$.

Définition 1 : *Affectation d'une variable*
On appelle affectation *d'une variable X_i, le fait d'attribuer à X_i une valeur de son domaine. L'affectation de la variable X_i à la valeur v_j est notée (X_i=v_j).*

Une *affectation complète* $\mathcal{A}=\{(X_1=v_1),...,(X_n=v_n)\}$ est une affectation de toutes les variables de $\mathcal{X}$. Lorsque qu'au moins une variable n'est pas affectée, on parlera d'*affectation partielle* (notée $\mathcal{A}_p$).

1.1.2 Contraintes

Soit $\mathcal{C}$ un ensemble contenant e contraintes. Chaque contrainte $c \in \mathcal{C}$ porte sur un ensemble de variables noté $\mathcal{X}_c$. Cet ensemble de variables est appelé *portée*[2] de la contrainte c.

Une contrainte est une relation qui impose des conditions sur les valeurs qui peuvent être affectées aux variables de sa portée. Ces restrictions peuvent être exprimées de manière symbolique (par exemple $X_1 < X_2$), dans ce cas on parle de *contrainte en intension*. Elles peuvent aussi être exprimées sous la forme d'un ensemble de *tuples autorisés*, c'est-à-dire l'ensemble des affectations des variables satisfaisant la contrainte. On parle dans ce cas de *contrainte en extension* ou de *contrainte table*.

Définition 2 : *Contrainte satisfaite*
Une contrainte c est dite satisfaite ssi *les variables $\mathcal{X}_c$ sont complètement instanciées et forment un tuple vérifiant c.*

Soit les deux variables X_1 et X_2 de domaines $D_{X_1} = D_{X_2} = \{1, 2\}$, la contrainte $X_1 \neq X_2$ est satisfaite si X_1 est affectée à la valeur 1 et X_2 à 2 (ou inversement).

[2]La portée d'une contrainte c est aussi appelée *scope* de c.

Un tuple autorisé est une affectation de toutes les variables de la portée d'une contrainte c satisfaisant celle-ci. Par opposition, un *tuple interdit* pour une contrainte c correspond à une affectation complète des variables de $\mathcal{X}_c$ ne satisfaisant pas c.

Les contraintes portant sur une variable sont dites unaires. Les variables portant sur deux variables, trois variables et n variables sont respectivement appelées binaires, ternaires et n-aires.

Définition 3 : *Contrainte globale*
Une contrainte globale *est une contrainte pouvant être utilisée avec différents nombres de variables dans sa portée. Une contrainte globale peut porter sur m variables comme sur n variables sans avoir besoin d'être redéfinie. Généralement, elles agglomèrent un ensemble de propriétés qu'il est plus agréable et efficace d'utiliser sous la forme d'une seule contrainte plutôt que d'une conjonction de contraintes équivalente.*

Par exemple, la contrainte globale `AllDifferent` [Rég94] impose à un ensemble de n variables de prendre des valeurs deux à deux différentes. Il est possible de retrouver la contrainte `AllDifferent` portant sur trois ou n variables en fonction du problème. Peu importe le nombre de variables dans sa portée, la contrainte `AllDifferent` modélise toujours le fait que toutes ces variables doivent prendre des valeurs deux à deux différentes.

Définition 4 : *Dureté d'une contrainte*
La dureté *d'une contrainte est le rapport entre le nombre de ses tuples interdits et la taille du produit cartésien des domaines de ses variables.*

Exemple 1 :
Soit les variables X_1 et X_2 avec pour domaines D_{X_1}=D_{X_2}=$\{1, 2, 3\}$ et la contrainte X_1+X_2>3. Les tuples interdits par cette contrainte sont {(1,1),(1,2),(2,1)}, ce qui représente 3 tuples interdits sur les 9 possibles, la dureté de cette contrainte est de 1/3.

Définition 5 : *Degré d'une variable*
Le degré *d'une variable X_i correspond au nombre de contraintes possédant X_i dans leur portée.*

1.1.3 Problème de satisfaction de contraintes

Une instance du problème de satisfaction de contraintes[3] (CSP) [Mon74, Mac77] est définie par un triplet $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, avec $\mathcal{X}$ l'ensemble des variables, $\mathcal{D}$ l'ensemble des domaines associés aux variables de $\mathcal{X}$ et $\mathcal{C}$ l'ensemble des contraintes. Un CSP dont la portée maximale des contraintes est 2 est dit binaire.

On peut représenter un CSP grâce à un *graphe* (resp. *hyper-graphe*) *de contraintes*. Dans ce graphe, chaque variable est représentée par un sommet, et chaque contrainte binaire (resp. n-aire) par une arête (resp. une hyper-arête) reliant les sommets associés aux variables de cette

[3]Par abus de langage, nous utiliserons le terme CSP pour désigner une instance du problème de satisfaction de contraintes.

contrainte.

Soit le réseau de contrainte $P=(\mathcal{X},\mathcal{D},\mathcal{C})$, avec $\mathcal{X}=\{X_1,X_2,X_3,X_4\}$, $\mathcal{D}=\{D_{X_1},D_{X_2},D_{X_3},D_{X_4}\}$, $D_{X_1}=D_{X_2}=\{1,2,3\}$, $D_{X_3}=\{2,3\}$, $D_{X_4}=\{2,3,4\}$ et $\mathcal{C}=\{\texttt{AllDifferent}(\{X_1,X_2,X_3\}), X_1+X_2 \leq 3, X_3 < X_4\}$. La figure 1.1 décrit le graphe de contraintes associé à P.

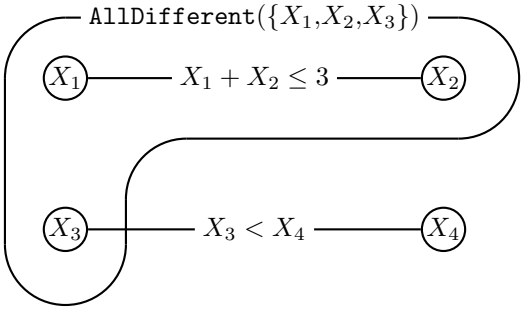


FIG. 1.1 – Réseau de contraintes du CSP P.

La notion de graphe de valeurs permet de représenter graphiquement les domaines des variables incluses dans une contrainte.

Définition 6 : *Graphe de valeurs*
Le graphe de valeurs *d'une contrainte c est un graphe bipartite $B=(\mathcal{X},\mathcal{Y},E)$. Les sommets de $\mathcal{X}$ représentent les variables de $\mathcal{X}_c$, et les sommets de $\mathcal{Y}$ les valeurs présentes dans les domaines des variables de $\mathcal{X}_c$. E est l'ensemble des arêtes tel que $\{X_i,v_j\}\in E$ ssi $X_i\in\mathcal{X}$ et $v_j\in D_{X_i}$.*

La figure 1.2 (resp. 1.3) représente le graphe de valeurs de la contrainte `AllDifferent` (resp. de la contrainte $X_3<X_4$) du réseau de contrainte P.

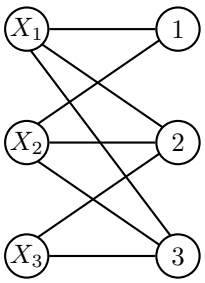


FIG. 1.2 – Graphe de valeurs de la contrainte $\texttt{AllDifferent}(\{X_1,X_2,X_3\})$.

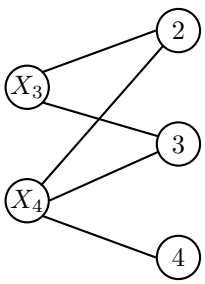


FIG. 1.3 – Graphe de valeurs de la contrainte $X_3<X_4$.

Définition 7 : *Solution d'un CSP*
Une solution d'un CSP P est une instanciation complète qui satisfait toutes les contraintes de P.

Le CSP P (Figure 1.1) possède deux solutions :
- $\{(X_1{=}1),(X_2{=}2),(X_3{=}3),(X_4{=}4)\}$;
- $\{(X_1{=}2),(X_2{=}1),(X_3{=}3),(X_4{=}4)\}$.

1.1.4 Exemple de modélisation : Le carré latin

Le problème du carré latin de dimension n consiste à remplir une matrice de dimension $n \times n$ avec des nombres compris entre 1 et n, de telle façon qu'aucun nombre ne soit répété sur une même colonne ou sur une même ligne. Généralement, un certain nombre de cellules sont déjà fixées.

Pour modéliser ce problème, on associe à chaque cellule (l,c) de la matrice une variable $X_{l,c}$ dont le domaine correspond aux valeurs que peut pendre cette cellule. Ainsi pour l'exemple présenté dans la figure 1.4, on aura un ensemble de 16 variables de domaine {1,2,3,4}.

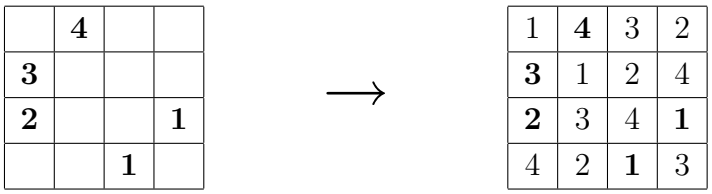


FIG. 1.4 – Exemple de carré latin de dimension 4.

Les contraintes du problème peuvent être divisées en trois ensembles :

- *les contraintes d'affectation* : pour toute cellule ayant déjà une valeur, on retrouve une contrainte unaire d'égalité affectant la variable correspondante à cette cellule. Par exemple pour la cellule (1,2), on a la contrainte $X_{1,2}$=4.
- *les contraintes de différence sur les lignes* : toutes les cellules d'une même ligne doivent prendre des valeurs différentes. On peut modéliser cela grâce à la contrainte globale `All-Different`. Par exemple, les cellules de la première ligne seront liées par la contrainte $\texttt{AllDifferent}(\{X_{1,1}, X_{1,2}, X_{1,3}, X_{1,4}\})$.
- *les contraintes de différence sur les colonnes* : de même que pour les lignes, pour chaque colonne on impose une contrainte `AllDifferent` portant sur toutes les cellules de cette colonne.

Modélisation 1 :

`// Variables et domaines`
$\mathcal{X} = \{X_{l,c} \mid l \in [1..4],\ c \in [1..4]\}$
$\forall l \in [1..4],\ \forall c \in [1..4],\ D_{X_{l,c}} = \{1, 2, 3, 4\}$
`// Contraintes d'affectation`
$X_{1,2} = 4$
$X_{2,1} = 3$
$X_{3,1} = 2$
$X_{3,4} = 1$
$X_{4,3} = 1$
`// Contraintes de différence (lignes)`
$\forall l \in [1..4],\ \texttt{AllDifferent}(\{X_{l,1}, X_{l,2}, X_{l,3}, X_{l,4}\})$
`// Contraintes de différence (colonnes)`
$\forall c \in [1..4],\ \texttt{AllDifferent}(\{X_{1,c}, X_{2,c}, X_{3,c}, X_{4,c}\})$

1.2 Cohérence d'un réseau de contraintes

Afin de réduire la taille de l'espace de recherche qu'il est nécessaire d'explorer pour trouver une solution, plusieurs niveaux de cohérence et mécanismes de filtrage ont été proposés. Ces mécanismes de filtrage permettent de retirer, des domaines des variables, les valeurs n'appartenant à aucune solution (du réseau de contraintes). Le nouveau réseau de contraintes, dit *filtré*, est équivalent au réseau initial.

Définition 8 : *Réseau de contraintes équivalents*
Deux réseaux de contraintes sont équivalents ssi *ils possèdent le même ensemble de solutions.*

Dans la suite de cette section, nous présentons la notion de cohérence d'arc, ainsi que plusieurs mécanismes de filtrage.

1.2.1 Cohérence d'arc dans les réseaux de contraintes binaires

La cohérence d'arc, définie sur un réseau de contraintes binaires, permet de filtrer les domaines des variables de manière locale à chaque contrainte, en exploitant la notion de *support*.

Définition 9 : *Support*
Soit une contrainte binaire c portant sur les variables X_i et X_j. La valeur (X_i,v_1) est support de la valeur (X_j,v_2) pour la contrainte c ssi *l'affectation partielle $\{(X_i=v_1),(X_j=v_2)\}$ satisfait c.*

La notion de support peut être représentée grâce au *graphe de micro-structures*. Le graphe de micro-structure est un graphe n-partite où chaque partie représente une variable, les sommets de chaque partie correspondent aux valeurs de la variable représentée, et les arêtes aux supports.

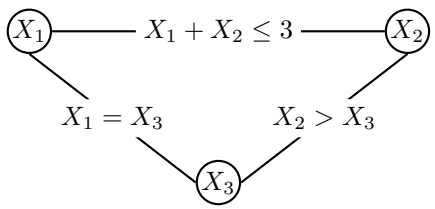


FIG. 1.5 – Graphe de contraintes

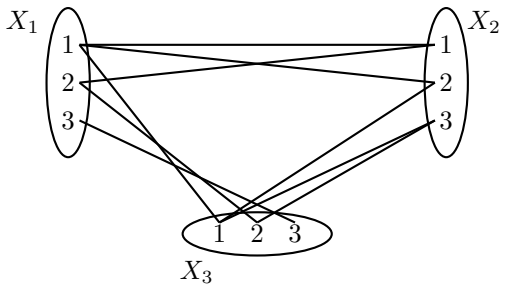


FIG. 1.6 – Graphe de micro-structures

Grâce à la notion de support, il est possible de savoir si une valeur donnée peut satisfaire une contrainte.

Définition 10 : *Viabilité d'une valeur (pour l'arc-cohérence)*
Une valeur (X_i,v_j) est viable ssi *pour toute contrainte binaire $c_{X_iX_k}$, (X_i,v_j) possède au moins un support dans le domaine de X_k.*

Si une valeur est non viable alors il n'existe pas d'instanciation complète utilisant cette valeur

et satisfaisant toutes les contraintes du problème. C'est pourquoi, il est possible de filtrer cette valeur tout en conservant toutes les solutions du problème initial. Dans l'exemple des figures 1.5 et 1.6, la valeur $(X_2, 1)$ ne possède pas de support pour la contrainte $X_2 > X_3$, cette valeur est donc non viable et peut être filtrée.

De plus, le fait de retirer une valeur du domaine d'une variable peut rendre non viable une autre valeur. Il est donc possible de *propager* les retraits. Toujours dans l'exemple précédent, le retrait de la valeur $(X_2, 1)$ rend la valeur $(X_1, 2)$ non viable car il s'agissait de son seul support pour la contrainte $X_1 + X_2 \leq 3$.

Définition 11 : *Cohérence d'arc d'un réseau de contraintes*
Un réseau de contraintes est arc-cohérent ssi *toutes ses valeurs sont viables et que le domaine d'aucune de ses variables n'est vide.*

De nombreux algorithmes de filtrage ont été proposés pour rendre un réseau de contraintes arc-cohérent. `AC-3` (cf. Algorithme 1) fut une des premières méthodes proposées [Mac77].

Le principe d'`AC-3` est de gérer les contraintes par l'intermédiaire d'un ensemble de contraintes à traiter Q. Initialement, cet ensemble contient tous les couples $(c_{X_iX_j}, X_i)$ et $(c_{X_iX_j}, X_j)$. Tant que Q n'est pas vide, le même traitement est effectué. On sélectionne le premier couple $(c_{X_iX_j}, X_i)$ de Q. À l'aide de la fonction `revise` le domaine de la variable X_i est filtré (dans le pire des cas en $O(d^2)$). Si D_{X_i} a été modifié par `revise` alors tous les couples $(c_{X_iX_k}, X_k)$, avec $k \neq j$, non présents dans Q sont insérés dans Q. À tout moment, si le domaine d'une variable est vidé alors `AC-3` retourne ECHEC car le réseau de contraintes ne possède pas de solution. Si Q devient vide alors l'algorithme s'arrête et le réseau de contraintes est arc-cohérent.

L'algorithme `AC-3` permet de maintenir la cohérence d'arc en $O(e \cdot d^3)$.

Plusieurs améliorations de cet algorithme ont été proposées. On peut notamment citer :

- `AC-4` [MH86] qui est la première méthode permettant de rendre arc-cohérent un réseau de contraintes en temps optimal dans le pire des cas, c'est-à-dire en $O(e \cdot d^2)$. Cette méthode propose de calculer dans un premier temps les supports de toutes les valeurs en $O(e \cdot d^2)$ et ensuite d'effectuer le filtrage en $O(e \cdot d^2)$.
- `AC-6` [BC93] qui permet de diminuer la complexité en moyenne en gérant de manière paresseuse les supports. En effet, au lieu de calculer l'ensemble de tous les supports comme `AC-4`, `AC-6` s'assure seulement qu'il en existe au moins un. Et à l'inverse d'`AC-3`, lorsqu'un support est retiré `AC-6` ne recherche pas à nouveau un support à partir de la première valeur du domaine mais depuis celle venant d'être retirée.
- `AC-2001` [BR01] améliore encore la propagation en stockant dans une structure de données additionnelle le dernier support et en testant en premier la présence de celui-ci.

1.2.2 Cohérence globale

Établir la cohérence globale pour une contrainte c revient à éliminer toutes les valeurs des domaines des variables de $\mathcal{X}_c$ qui ne peuvent pas être étendues en un tuple autorisé par c.

Algorithme 1 : `AC-3`

```
1  fonction revise(c_{X_iX_j}, X_i)
2  début
3      change ← faux
4      pour chaque a ∈ D_{X_i} faire
5          si ∄b ∈ D_{X_j} t.q. b soit un support de a sur c_ij alors
6              D_{X_i} ← D_{X_i}\{a}
7              change ← vrai
8              si D_{X_i} = ∅ alors retourner ECHEC
9      retourner change
10 fin

11 procédure AC-3(𝒳, 𝒞)
12 début
13     // initialisation
14     Soit Q un ensemble vide
15     pour chaque c_{X_iX_j} ∈ 𝒞 faire
16         Q ← Q ∪ {(c_{X_iX_j}, X_i)} ∪ {(c_{X_iX_j}, X_j)}
17     // propagation
18     tant que Q ≠ ∅ faire
19         Extraire (c_{X_iX_j}, X_i) de Q
20         si revise(c_{X_iX_j}, X_i) alors
21             Q ← Q ∪ {(c_{X_iX_k}, X_k), k ≠ j}
22 fin
```

Définition 12 : ***Cohérence globale***

Soit une contrainte $c(X_1, ..., X_n)$, cette contrainte est globalement cohérente ssi *pour toute valeur (X_i, v_j) il existe une instanciation complète $\mathcal{A} \in D_1 \times ... \times D_{i-1} \times \{v_j\} \times D_{i+1} \times ... \times D_n$ telle que c soit satisfaite.*

Par exemple, la contrainte `AllDifferent`$(\{X_1, X_2, X_3\})$ avec pour domaines D_{X_1}=$\{1, 2\}$, D_{X_2}=$\{2\}$, D_{X_3}=$\{3\}$ n'est pas globalement cohérente car la valeur $(X_1, 2)$ ne peut être étendue à une instanciation complète satisfaisant la contrainte `AllDifferent`.

La cohérence globale est aussi désignée par les termes GAC (*generalized arc consistency*), ou cohérence d'hyper-arc.

Définition 13 : ***Viabilité d'une valeur (pour la cohérence globale)***

Une valeur (X_i,v_j) est viable ssi *pour toute contrainte c, ayant X_i dans sa portée, il existe une instanciation complète des variable de $\mathcal{X}_c$ étendant l'affectation $(X_i = v_j)$ et satisfaisant la contrainte c.*

Il est à noter que pour une contrainte binaire, la cohérence globale est équivalente à la cohérence d'arc.

Remarque 1 : Pour simplifier les discussions à venir, lorsque toutes les valeurs des domaines de toutes les variables d'une contrainte (globale) c peuvent être étendues en une instanciation complète satisfaisant c alors nous dirons que la contrainte c est globalement cohérente.

1.3 Recherche de solution

Dans le cadre des problèmes de satisfaction de contraintes, on peut distinguer trois types de problèmes fondamentaux :

- l'existence d'une solution ;
- la recherche d'une solution ;
- le comptage et/ou la recherche de toutes les solutions d'un problème.

Le problème de l'existence d'une solution est un problème NP-Complet. Les deux problèmes de la recherche d'une solution et du comptage de toutes les solutions sont des problèmes de difficulté plus élevée.

Dans la suite de cette thèse, nous nous intéresserons essentiellement au problème de la recherche d'une solution. Nous utiliserons l'exemple suivant :

Exemple 2 :
Le but est de trouver une solution au problème du carré latin de dimension 3 de la figure 1.7.

1		
	1	

FIG. 1.7 – Carré latin de dimension 3.

1.3.1 Arbre de recherche

i) Structure d'un arbre de recherche

L'espace de recherche peut être représenté par un arbre : à chaque nœud de l'arbre (ou point de choix) correspond une variable et à chaque branche une décision. Suivant la nature du point de choix et le nombre de branches sortant de celui-ci, nous parlons de branchement :

- d-aire : chaque branche issue d'un même nœud représente une des différentes affectations possibles de la variable ;
- binaire : à chaque point de choix, deux branches sont associées. Sur la première branche, le choix d'affecter la valeur v_j à la variable X_i est fait $(X_i = v_j)$, et sur la seconde le choix de ne pas affecter cette variable à cette valeur $(X_i \neq v_j)$;

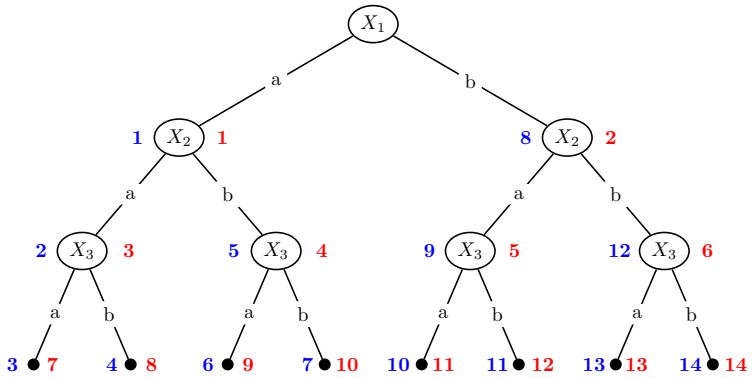


FIG. 1.8 – Les valeurs à gauche des nœuds représentent l'ordre de parcours des nœuds selon la stratégie en profondeur d'abord et celles de droite en largeur d'abord.

– par séparation de domaine : chaque décision réduit le domaine de la variable (sans forcément l'affecter). Par exemple, pour $D_{X_1}=\{1, ..., 10\}$, la contrainte $X_1 > 4$ peut être ajoutée sur une branche et $X_1 \leq 4$ sur une autre.

Chaque feuille de profondeur n représente une instanciation complète résultant des instanciations faites le long de la branche menant à cette feuille. Si cette instanciation complète satisfait toutes les contraintes du problème alors il s'agit d'une solution.

ii) Stratégies d'exploration

Il existe plusieurs stratégies pour explorer un arbre de recherche :

– *le parcours en profondeur d'abord* (depth first) : cette stratégie favorise la descente dans l'arbre de recherche en explorant en premier les nœuds les plus à gauche dans l'arbre de recherche.

– *le parcours en largeur d'abord* (breadth first) : l'arbre est exploré par niveau de la gauche vers la droite.

– *le parcours en utilisant le meilleur d'abord* (best first) : cette stratégie utilise la gestion d'une frontière entre les nœuds explorés et ceux non explorés (cette frontière peut être représentée par une file à priorités). Au moment de choisir un nœud à explorer, le nœud de la frontière qui est le plus prometteur selon un critère d'évaluation est sélectionné. Une fois un nœud traité, tous ses fils sont ajoutés à la frontière.

La figure 1.8 présente sur un exemple l'ordre d'exploration des nœuds d'un arbre de recherche selon les stratégies en profondeur d'abord, et en largeur d'abord.

Dans la suite de cette section, nous utiliserons la stratégie de parcours en profondeur d'abord[4]. Sur l'exemple du carré latin de dimension 3 de la figure 1.7, plus de 1400 branches doivent être explorées avant d'atteindre une première solution, soit environ 65% de l'espace de recherche.

[4]Nous utiliserons l'ordre lexicographique pour ordonner les variables et les valeurs.

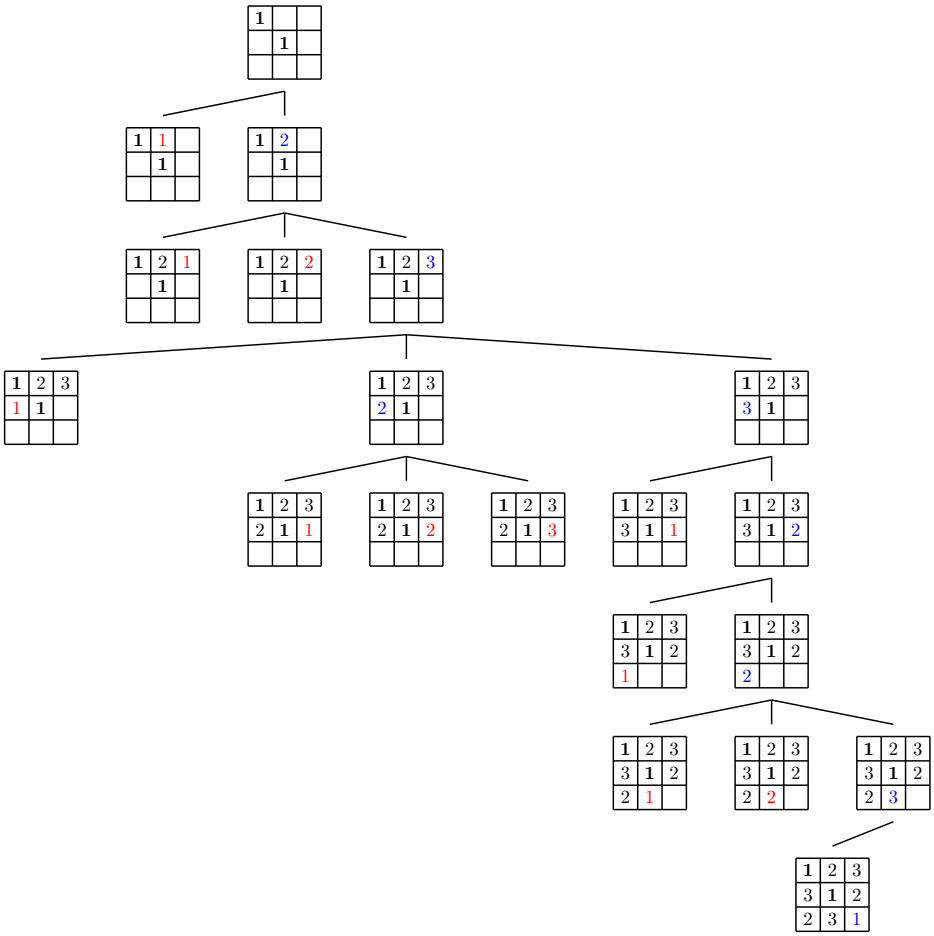


FIG. 1.9 – Arbre de recherche exploré par l'algorithme `backtrack`

1.3.2 Algorithme de `backtrack`

Le principe de l'algorithme de `backtrack` (retour arrière) est le suivant : l'arbre de recherche est exploré en profondeur d'abord, et à chaque nœud de l'arbre est associée une affectation partielle $\mathcal{A}_p$ résultant des affectations depuis la racine de l'arbre à celui-ci. À chaque nœud, l'algorithme étend l'affectation partielle $\mathcal{A}_p$ en affectant une variable X_{p+1} (à une valeur de son domaine). Si la nouvelle affectation est localement cohérente alors l'algorithme continue en affectant la variable X_{p+2}. Dans le cas contraire, il revient sur la dernière affectation de X_{p+1} et teste une nouvelle valeur de son domaine. Si toutes les valeurs de $D_{X_{p+1}}$ ont été envisagées, l'algorithme effectue un retour arrière (un *backtrack*) sur la variable X_p.

Au début de l'algorithme, l'instanciation partielle $\mathcal{A}_p$ est vide. L'algorithme s'arrête soit après avoir atteint une feuille de l'arbre de recherche ($\mathcal{A}_p$ est alors complète et correspond à une solution), soit lorsque l'espace de recherche a été totalement exploré (l'instance ne possède pas de solution).

La complexité dans le pire cas de l'algorithme `backtrack` est en $O(d^n)$, avec n le nombre de variables et d la taille du plus grand domaine. En effet, dans le pire des cas l'arbre de recherche

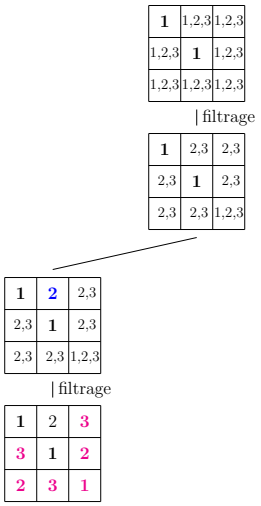


FIG. 1.10 – Arbre exploré par l'algorithme `MAC`

est exploré en entier.

Sur l'exemple 2, une première solution est trouvée après l'exploration de 12 branches (voir figure 1.9).

1.3.3 Algorithme de `backtrack` avec maintien de cohérence

Pour trouver une solution à un réseau de contraintes, on utilise généralement l'algorithme de `backtrack` auquel sont ajoutés des mécanismes de filtrage (cf. section 1.2). Pour réduire l'espace de recherche, à chaque nœud de l'arbre, les valeurs non viables des variables sont filtrées. Ainsi, un grand nombre de branches sans solutions ne seront pas explorées. On peut notamment citer l'algorithme `MAC` (Maintien de Cohérence d'Arc) qui établit la propriété de cohérence d'arc à chaque nœud de l'arbre de recherche.

La figure 1.10 montre l'arbre de recherche exploré par `MAC` sur le problème de l'exemple 2.

1.3.4 Heuristiques

Le principe d'une heuristique est d'accélérer la recherche en définissant un ordre d'exploration de l'arbre de recherche de façon à diminuer l'espace qu'il est nécessaire de parcourir avant d'atteindre une solution.

Pour un problème donné, les heuristiques de choix de variables et de valeurs jouent des rôles différents. L'heuristique de choix de variables détermine un ensemble de sous-arbres de recherche à explorer, et l'heuristique de choix de valeurs détermine l'ordre de résolution de ces sous-arbres.

Les heuristiques peuvent être de deux natures :

- *statique* : les choix sont effectués avant la recherche et sont figés ;
- *dynamique* : les choix dépendent du sous-arbre à explorer (par exemple, nombre de variables affectées, tailles des domaines, choix précédents, ...) et sont effectués à chaque nœud pendant la recherche.

i) Heuristiques de choix de variables

Le but d'une heuristique de choix de variables est de réduire la taille de l'espace de recherche, généralement en provoquant le plus rapidement possible un échec.

De nombreuses heuristiques de choix de variables ont été proposées. Parmi celles-ci, on retrouve notamment :

- *domaine minimum* (dom) [HE80] : cette heuristique favorise les variables avec les domaines les plus petits ;
- *degré maximum* (deg) [DM89] : les variables qui ont un degré élevé provoquent généralement plus de retraits, cette heuristique favorise donc les variables dont les degrés sont les plus élevés ;
- *domaine/degré* (dom/deg) [BR96] : cette heuristique propose de tirer parti des informations des deux heuristiques précédentes et favorise les variables dont le ratio de la taille de leur domaine par rapport à leur degré est le plus faible ;
- *domaine/degré futur* (dom/fdeg) [SG98] : il s'agit d'une variante de dom/deg dans laquelle le degré futur d'une variable est utilisé à la place du degré. Le degré futur d'une variable correspond au nombre de contraintes portant sur une variable et dont au moins une variable n'est pas affectée.

ii) Heuristiques de choix de valeurs

Les heuristiques de choix de valeurs cherchent à aiguiller la recherche vers les branches les plus prometteuses de l'arbre de recherche.

Parmi ces heuristiques de choix de valeurs, on retrouve :

- *lexicographique* (lex) : les valeurs sont simplement ordonnées selon l'ordre de leur domaine.
- *basée « look-ahead »* [FD95] : cette heuristique classe les valeurs selon l'ordre croissant du nombre de retraits provoqués par l'affectation de celle-ci.
- *basée sur les explications* [CJ06] : cette heuristique utilise les informations liées au retrait d'une valeur (c'est-à-dire aux affectations justifiant ce retrait).
- *le meilleur d'abord* (best first) : cette heuristique classe les valeurs grâce au score d'une fonction évaluant les chances d'obtenir une solution dans son sous-arbre. Par exemple, cette fonction peut être le nombre de support d'une valeur, le nombre d'utilisation de celle-ci, etc.

Un lecteur intéressé peut se référer au mémoire de thèse de Nicolas Levasseur [Lev08].

1.3.5 Limited discrepancy search : `LDS`

`LDS` a été introduite par William D. Harvey et Matthew L. Ginsberg [HG95]. Soit h une heuristique de choix de valeurs dans laquelle on a une grande confiance. Le principe de `LDS` est de suivre l'heuristique h lors du parcours de l'arbre de recherche, mais en permettant seulement un petit écart (δ) par rapport aux choix de h. On s'autorise donc δ écarts (ou *discrepancies*) à l'heuristique h. Pour un nombre maximal δ_{max} d'écarts autorisés, `LDS` explore l'arbre de recherche

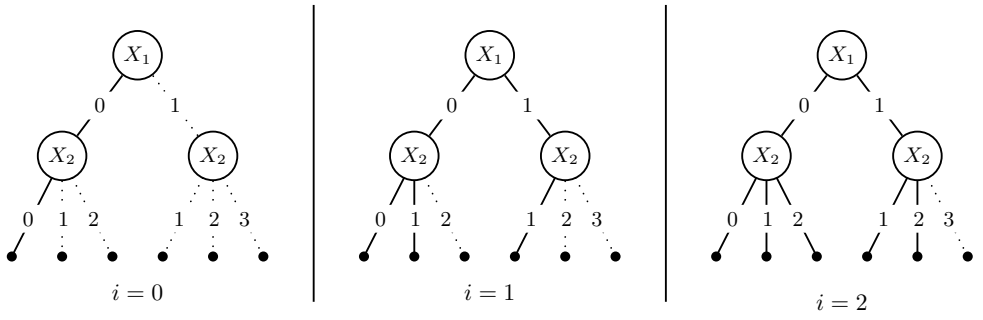


FIG. 1.11 – Arbres de recherche explorés par `LDS` avec $\delta_{max} = 2$.

de manière itérative selon un nombre croissant d'écarts allant de i=0 à i=δ_{max}. À chaque itération i est incrémenté de 1.

Dans la suite, nous ne considérons que des arbres d-aires. Dans un tel cas, l'heuristique h va ordonner toutes les valeurs du domaine d'une variable. La première valeur, selon l'ordre donné par h, n'engendre aucune augmentation de δ. La deuxième valeur augmente la valeur de δ de 1 et la troisième de 2, etc.

L'intérêt de cette méthode est d'explorer seulement les parties les plus intéressantes de l'arbre de recherche (selon les heuristiques) et non son intégralité comme le fait `MAC`.

La figure 1.11 représente les branches de l'arbre de recherche exploré (en trait plein) lors des différentes itérations de `LDS` (avec δ_{max}=2).

1.3.6 Méthodes de recherche locale

Les méthodes de recherche locale, contrairement aux méthodes complètes, effectuent des déplacements dans l'espace de recherche des solutions. À chaque déplacement (ou mouvement), l'algorithme tente d'améliorer la solution courante s en sélectionnant heuristiquement une solution s' dans son voisinage $\mathcal{N}(s)$. Le processus s'arrête lorsque la solution courante n'a pas pu être améliorée pendant un certain nombre de mouvements (fixé au départ) ou que le temps maximal alloué à la recherche est atteint.

Bien que ces méthodes ne soient pas complètes et n'offrent aucune garantie d'optimalité, elles produisent très souvent des solutions de bonne qualité en des temps de calcul « raisonnables ».

1.4 Notions de graphes et graphes orientés

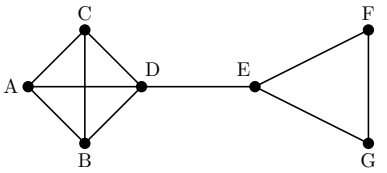
1.4.1 Définitions générales

Définition 14 : *Graphe*

Un graphe *G=(V,E) est défini par :*

- *un ensemble de sommets V ;*
- *un ensemble d'arêtes E reliant deux sommets mutuellement.*

On désignera par $\{v_i,v_j\}$ l'arête reliant les sommets v_i et v_j. On dira que deux sommets sont *connectés* si et seulement si une arête relie ces sommets.

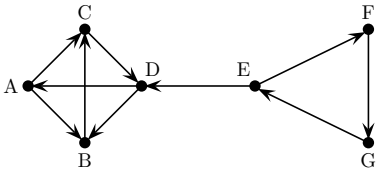


G=(V,E) avec
V={A,B,C,D,E,F,G}
et
E ={{A,B},{A,C},{A,D},{B,C},{B,D},
{C,D},{D,E},{E,F},{E,G},{F,G}}

FIG. 1.12 – Exemple de graphe.

Définition 15 : ***Graphe orienté***
Un graphe orienté *G=(V,A) est défini par :*
- *un ensemble de sommets V ;*
- *un ensemble d'arcs A reliant un sommet à un autre.*

On notera $(v_i \rightarrow v_j)$ l'arc reliant le sommet v_i au sommet v_j.



G=(V,E) avec
V={A,B,C,D,E,F,G}
et
A ={(A→B),(A→C),(B→C),(C→D),(D→A),
(D→B),(E→D),(E→F),(F→G),(G→E)}

FIG. 1.13 – Exemple de graphe orienté.

Les figures 1.12 et 1.13 décrivent respectivement un exemple de graphe non orienté et un exemple de graphe orienté. Il est possible de transformer un graphe non orienté en un graphe orienté équivalent en substituant deux arcs à chaque arête. L'arête $\{v_i,v_j\}$ sera remplacée par les arcs $(v_i \rightarrow v_j)$ et $(v_j \rightarrow v_i)$.

Dans un graphe orienté, nous désignerons par $\delta^+(v_i)$ (resp. $\delta^-(v_i)$) le nombre d'arcs entrants (resp. sortants) du sommet v_i.

Définition 16 : ***Graphe partiel***
Soit un graphe G=(V,E), on dira que G′=(V,E′) est un graphe partiel *de G* ssi $E' \subseteq E$.

Définition 17 : ***Sous-graphe***
Soit un graphe G=(V,E), le sous-graphe engendré par $V' \subseteq V$ *est le graphe G′=(V′,E′) tel que* $E' \subseteq E$ *soit l'ensemble des arêtes de G ayant leurs deux extrémités dans* V'.

Si le graphe est orienté on parlera alors de *sous-graphe induit*.

Définition 18 : ***Graphe complet***
Un graphe G=(V,E) est complet ssi *pour toute paire de sommets* v_i *et* v_j *l'arête* $\{v_i,v_j\}$ *appartient à E.*

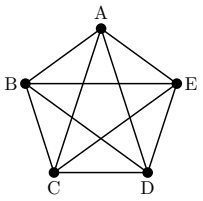


FIG. 1.14 – Graphe complet à cinq sommets.

Définition 19 : *Clique*
Soit un graphe G=(V,E) (non orienté), une clique *de G est un sous-graphe complet de G. Une clique est dite* maximale *si elle n'est contenue dans aucune autre clique. La taille d'une clique est son nombre de sommets.*

La figure 1.14 représente un graphe complet à cinq sommets, ce graphe est appelé K_5. Dans la figure 1.12, la plus grande clique est composée des sommets A, B, C et D.

Définition 20 : *Graphe k-partite*
Un graphe est k-partite ssi *l'ensemble de ses sommets est décomposable en k ensembles disjoints tels que toute paire de sommets d'un même ensemble ne soit pas reliée par une arête.*

Par la suite, nous nous intéresserons plus particulièrement aux *graphes bipartites* (ou 2-partites).

1.4.2 Chaînes et chemins

Définition 21 : *Chaîne*
Soit un graphe G=(V,E), une chaîne est une succession d'arêtes telle que on ait $\{v_1,v_2\}$, $\{v_2,v_3\},\ldots,\{v_n,v_{n+1}\}$.

Si aucune confusion n'est possible, alors une chaîne peut être désignée par la succession des sommets qu'elle parcourt.

Définition 22 : *Chaîne élémentaire*
Une chaîne est dite élémentaire ssi *elle ne passe pas plus d'une fois par un même sommet.*

Définition 23 : *Cycle*
Une chaîne dont les deux extrémités coïncident est appelée un cycle.

Dans le cas d'un graphe orienté, une chaîne (resp. un cycle) est désignée par le terme de *chemin* (resp. de *circuit*).

Définition 24 : *Plus court chemin*
Le plus court chemin entre deux sommets (resp. la plus courte chaîne) est le chemin dont la somme des poids des arcs (resp. des arêtes) utilisés pour relier ces deux sommets est minimale.

Dans le cas d'un graphe non pondéré, on considère que chaque arête (resp. arc) est de poids 1. Nous désignerons par $weight(p)$ la somme des poids des arêtes (resp. arcs) participant à la chaîne p (resp. au chemin p).

Définition 25 : *Plus court cycle*
Le plus court cycle non trivial (resp. circuit) est le plus court chemin (resp. la plus courte chaîne) dont la somme des poids des arcs (resp. des arêtes) utilisés pour relier un sommet à lui-même.

La recherche du plus court chemin peut être résolue de plusieurs façons en fonction du graphe :

- Si le graphe est acyclique alors la recherche du plus court chemin peut être effectuée en temps $O(|E|)$ grâce à un parcours en largeur d'abord du graphe.
- Si le graphe ne contient pas d'arcs de coût négatif alors on peut utiliser l'algorithme de `Dijkstra` [Dij59] en temps $O(|E| + |V| \cdot log(|V|))$.

1.4.3 Voisinage d'un sommet et composante connexe

Définition 26 : *Voisinage d'un sommet*
Le voisinage d'un sommet v, noté $\Gamma(v)$, est l'ensemble des sommets reliées par une arête à v (ou un arc sortant de v).

Dans le graphe de la figure 1.12, on a Γ(A)={B,C,D}. Et dans le cas du graphe orienté de la figure 1.13, on a Γ(A)={B,C}.

Définition 27 : *Composante connexe*
Une composante connexe, dans un graphe (non orienté) quelconque, est un ensemble maximal de sommets tel que pour toute paire de sommets v_i et v_j, appartenant à cet ensemble, il existe une chaîne reliant v_i et v_j.

Définition 28 : *Composante fortement connexe*
Une composante fortement connexe, dans un graphe orienté quelconque, est un ensemble maximal de sommets tel que pour toute paire de sommets v_i et v_j, appartenant à cet ensemble, il existe un chemin reliant v_i à v_j et un chemin reliant v_j à v_i.

Le graphe de la figure 1.12 est constitué d'une seule composante connexe. Le graphe orienté de la figure 1.13 est composé de deux composantes fortement connexes {A,B,C,D} et {E,F,G}.

Les composantes fortement connexes peuvent être calculées en temps $O(|V|+|E|)$ grâce à l'algorithme de `Tarjan` [Tar72].

1.4.4 Couplage

Définition 29 : *Couplage*
Etant donné un graphe G=(V,E), un couplage *M est un sous-ensemble d'arêtes telle qu'aucune paire d'arêtes n'ait un sommet en commun.*

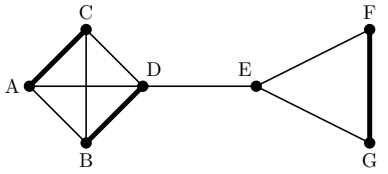


FIG. 1.15 – Exemple de couplage maximal

À partir de cette définition, on peut définir les notions portant sur la taille d'un couplage et sur les sommets hors du couplage.

Définition 30 : *Cardinalité d'un couplage*
La cardinalité d'un couplage M *(ou encore sa taille) est le nombre d'arêtes comprises dans* M.

Définition 31 : *Couplage maximal*
Un couplage M *de cardinalité maximale est appelé* couplage maximal.

Définition 32 : *Sommet libre*
Soit un graphe $G=(V,E)$ *et un couplage* M, *un* sommet libre *pour* M *est un sommet qui n'est une extrémité pour aucune arête de* M.

Le couplage $M = \{\{A, C\}, \{B, D\}, \{F, G\}\}$ de la figure 1.15 (arcs en gras) est maximal avec une cardinalité de 3. Le sommet E est le seul sommet libre de ce couplage.

Définition 33 : *Couplage couvrant*
Soit un graphe $G=(V,E)$, $V' \subseteq V$ *et un couplage* M, M couvre V' ssi *aucun sommet* $v \in V'$ *n'est libre pour* M.

Définition 34 : *Chaîne alternée et cycle alterné*
Soit un graphe $G=(V,E)$ *et un couplage* M, *une chaîne alternée (resp. un cycle alterné) est une chaîne (resp. un cycle) élémentaire dont les arêtes sont alternativement prises dans* M *et dans* $E \backslash M$.

Dans le couplage présenté dans la figure 1.15, la chaîne $\{C, A, B, D, E\}$ est une chaîne alternée et le cycle $\{A, B, D, C, A\}$ est lui aussi alterné.

Dans le cas d'un graphe bipartite, il est possible de calculer un couplage maximal en $O(\sqrt{|V|} \times \ |E|)$ grâce à l'algorithme d'`Hopcroft & Karp` [HK73]. Dans le cas d'un graphe quelconque, un couplage maximal peut être déterminé en $O(|V| \times |E|)$ en utilisant l'algorithme d'`Edmonds` [Edm65].

1.5 Notions de réseaux et flots

1.5.1 Réseaux et flots

Définition 35 : *Réseau*

Un réseau $R = (V, A)$ *est un graphe orienté muni de deux sommets particuliers, la source, notée* s*, et le puits, noté* t*. À chaque arc on associe :*

- *une fonction de demande* $d : A \to \mathbb{N}^+$ *qui correspond à la quantité de flot qui doit passer par l'arc* a *;*
- *une fonction de capacité* $c : A \to \mathbb{N}^+$ *qui correspond à la quantité maximale de flot qui peut passer par* a*.*

Une fonction w *associe à chaque arc* a *un poids* $w(a)$*. Par défaut, le poids d'un arc est considéré comme nul.*

Les arcs seront étiquetés de la façon suivante : (demande,capacité) w=poids. Il est à noter que lorsqu'un arc a n'est pas pondéré alors $w(a)$ est omis. Lorsque d'une unité de flot circulera dans un arc, celui-ci sera représenté en gras. Pour deux unités de flot, cet arc sera représenté en gras et possédera deux têtes de flèches. Finalement pour toute valeur u de flot supérieure à 2, l'arc sera représenté en gras et le nombre d'unité de flot passant par celui sera précisé par la notation f=u.

Le problème principal associé aux réseaux est la recherche d'un flot.

Définition 36 : *Flot*

Un flot *dans un réseau* $R = (V, A)$ *est une fonction* $f : A \to \mathbb{N}^+$ *telle que :*

- *Pour tout arc* $a \in A$*,* $d(a) \leq f(a) \leq c(a)$*.*
- *Pour tout sommet* $v \in V$*,* f *doit respecter la loi de conservation :*

$$\sum_{(u \to v) \in A} f(u \to v) = \sum_{(v \to u) \in A} f(v \to u)$$

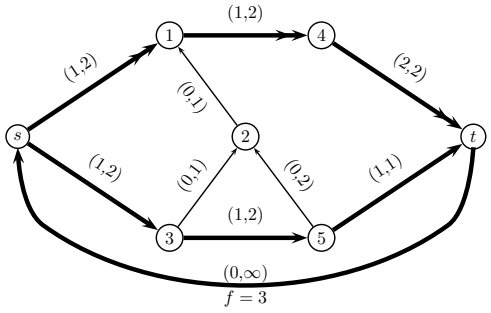


FIG. 1.16 – Exemple de flot faisable.

Afin que la loi de conservation soit vérifiée en tout sommet du réseau, on ajoute un arc ($t \to s$) avec une demande nulle, une capacité infinie et un poids nul[5]. La *valeur du flot* correspond au

[5]Dans la plupart des figures suivantes, l'arc ($t \to s$) ne sera pas représenté.

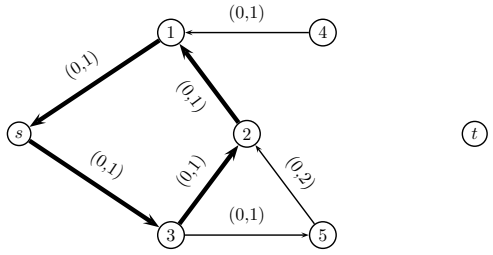


FIG. 1.17 – Réseau résiduel associé au flot de la figure 1.16.

nombre d'unités de flot entrant en t. La figure 1.16 représente un flot de valeur 3.

Définition 37 : *Poids d'un flot*
Soit un réseau $R = (V, A)$, f un flot dans ce réseau, et $w(a)$ le poids associé à l'arc a. Le poids *de f, noté $weight(f)$, est défini par :*

$$weight(f) = \sum_{a \in A} f(a) \times w(a)$$

L'algorithme de `Ford & Fulkerson` [FF56] permet de calculer un flot faisable (de poids minimal) dans un réseau en temps $O(K \times SP)$ avec K la valeur du flot et SP le temps nécessaire au calcul d'un plus court chemin.

1.5.2 Réseau résiduel et flot

Le *réseau résiduel* d'un flot correspond à l'état du réseau après le passage de ce flot ; c'est-à-dire à ce même réseau dans lequel les arcs ont été modifiés pour tenir compte des demandes restantes et des capacités encore disponibles après le passage de ce flot.

Il existe deux transformations possibles pour un arc donné ($a \rightarrow b$) de poids w, pour chaque unité de flot passe par celui-ci :

- si la demande n'est pas satisfaite : dans ce cas on diminue à la fois la demande et la capacité de 1.
- si la demande est déjà satisfaite : dans ce cas on diminue la capacité de l'arc ($a \rightarrow b$) de 1 et on ajoute un arc inverse ($b \rightarrow a$) avec $d = 0$, $c = 1$ et de poids[6] $-w$.

La première transformation représente le fait que lorsque qu'une unité de flot passe par un arc a alors la demande et la capacité de a sont diminuées de 1, car consommées par cette unité de flot. La seconde transformation permet de modéliser le fait que si la demande de a est déjà satisfaite alors l'unité de flot passant par a est "superflue" et peut donc être réutilisée pour satisfaire la demande d'un autre arc a'. Cette transformation est illustrée par la figure 1.17.

[6]Dans le cas où il existe déjà un arc ($b \rightarrow a$) de poids $-w$ alors la capacité de cet arc est incrémentée de 1.

i) Le cas non pondéré

Dans le cas où les arcs du réseau ne sont pas pondérés, la propriété suivante permet à partir d'un premier flot de déterminer s'il existe un flot passant par un arc donné.

Propriété 1 :
Soit un réseau R et f un flot faisable dans R, il est possible de calculer un flot faisable f' passant par l'arc $(u \rightarrow v)$ en modifiant f le long d'un circuit passant par $(u \rightarrow v)$ dans le réseau résiduel de f.

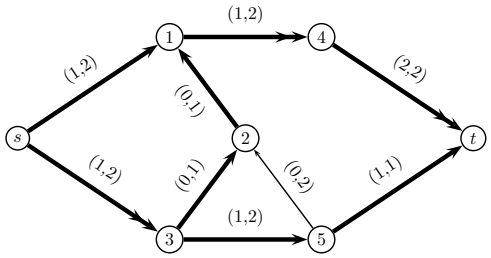


FIG. 1.18 – Exemple de flot faisable utilisant l'arc $(2 \rightarrow 1)$.

Remarque 2 : Dans la plus part des cas, un circuit est calculé via la recherche d'un plus court chemin reliant v à u.

À partir du flot f de la figure 1.16, il est possible de calculer un flot faisable utilisant l'arc $(2 \rightarrow 1)$ en calculant un chemin reliant 2 à 3 dans le réseau résiduel de f (cf. Figure 1.17). La figure 1.18 montre le nouveau flot faisable f' obtenu grâce au chemin $(2, 1, s, 3)$.

ii) Cas pondéré

La propriété 1 peut être étendue dans le cas pondéré en la propriété suivante :

Propriété 2 :
Soit un réseau R et f un flot faisable de poids minimal dans R, il est possible de calculer un flot faisable f' passant par l'arc $(u \rightarrow v)$, tel que f' soit un flot de poids minimal exploitant $(u \rightarrow v)$, en modifiant f le long du court circuit de plus petit poids passant par $(u \rightarrow v)$ dans le réseau résiduel de f.

La figure 1.19 représente un flot faisable de poids minimal dans un réseau pondéré. À partir de ce flot il est possible de calculer son réseau résiduel (Figure 1.20). Pour connaître le flot faisable de plus petit poids utilisant l'arc $(2 \rightarrow 1)$, il est nécessaire de calculer le circuit de plus petit poids dans ce réseau résiduel. Le circuit $(s, 3, 5, 2, 1)$ correspond à ce circuit est de poids 10. Ainsi en modifiant le flot en fonction de ce circuit, il est possible de déterminer le flot faisable de poids minimal exploitant l'arc $(2 \rightarrow 1)$ et celui-ci est de poids 31 (21 + 10). La figure 1.21 illustre ce flot de poids minimal passant par $(2 \rightarrow 1)$.

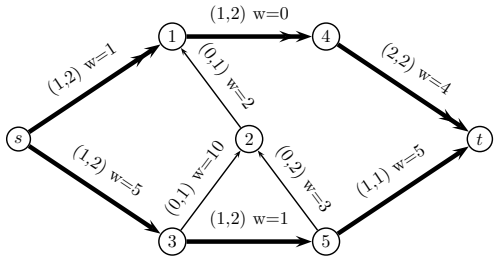


FIG. 1.19 – Flot faisable de poids minimal dans un réseau pondéré.

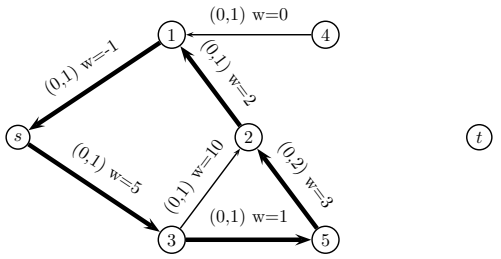


FIG. 1.20 – Réseau résiduel associé au flot de la figure 1.19.

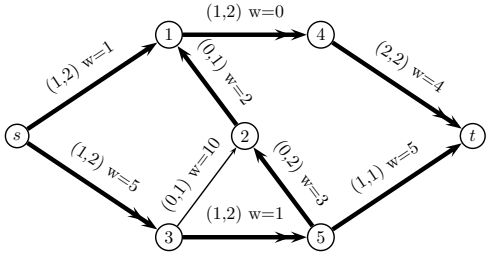


FIG. 1.21 – Flot faisable de poids minimal utilisant l'arc $(2 \to 1)$.

Remarque 3 : Par abus de langage, nous utiliserons le terme de cycle pour désigner les circuits.

1.6 Automates

Un automate est un graphe orienté particulier permettant, notamment, de déterminer si un mot donné appartient à un langage (décrit par l'automate).

Définition 38 : ***Automate fini déterministe***
Un automate fini déterministe *(DFA)* Π *est un quintuplet* $(Q, \Sigma, \delta, q_0, F)$ *avec* Q *un ensemble fini d'états,* Σ *un alphabet,* $\delta : Q \times \Sigma \to Q$ *un ensemble de transitions,* q_0 *un état initial et* $F \subseteq Q$ *un ensemble d'états finaux.*

Un automate est *complet ssi* pour tout couple $(q,a) \in Q \times \Sigma$ il existe une transition $\delta(q, a) = q'$ (avec $q' \in Q$).

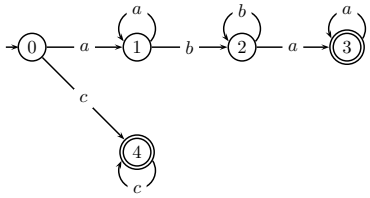


FIG. 1.22 – Exemple d'automate.

Un mot est *reconnu* par un automate Π *ssi* sa lecture par Π, partant de l'état initial, s'achève

sur un état final. Par extension, le langage reconnu par un automate est constitué de tous les mots reconnus par celui-ci. Il est à noter qu'une expression régulière peut être transformée en temps polynomial en un automate déterministe fini la reconnaissant [CP97].

L'automate de la figure 1.22 permet de reconnaître l'expression régulière : $(aa^{\star}bb^{\star}aa^{\star}) \mid cc^{\star}$. Cet automate n'est pas complet.

1.7 Conclusion

Dans ce chapitre, nous avons présenté les notions de base qui nous seront utiles par la suite :

- les *problèmes de satisfaction de contraintes* qui permettent de modéliser de nombreux problèmes à l'aide de variables possédant des domaines devant vérifier des contraintes, leur résolution peut se faire en utilisant une recherche arborescente ;
- la *théorie des graphes* (au sens général incluant les réseaux, les flots ou encore les automates) qui propose de résoudre des problèmes très spécifiques grâce à des algorithmes très performants.

Dans le chapitre suivant, nous présenterons trois contraintes globales :

- la contrainte globale `AllDifferent` qui est liée à la recherche de couplage maximal ;
- la contrainte de cardinalité globale (`Gcc`) qui utilise le calcul de flot ;
- la contrainte globale `Regular` qui exploite la théorie des langages et des automates.

Chapitre 2

Contraintes globales

Sommaire

De très nombreuses contraintes globales ont été développées depuis une vingtaine d'années (voir le catalogue des contraintes globales [BCDP07, BD09]). Dans ce chapitre, nous avons choisi de présenter plus en détail trois d'entre elles :

- `AllDifferent` qui impose qu'un ensemble de variables prenne des valeurs deux à deux différentes [Rég94],
- `Gcc` qui contraint le nombre d'occurrences de certaines valeurs dans une solution à respecter des bornes inférieures et des bornes supérieures [Rég96],
- `Regular` qui oblige une séquence de variables à former un mot reconnu par un automate déterministe fini [Pes04].

Notre choix s'est porté sur ces trois contraintes car elles figurent parmi les contraintes globales les plus connues et les plus utilisées dans le développement d'applications. Il s'agit aussi des contraintes sur lesquelles ont porté les tous premiers travaux sur la relaxation de contraintes globales [PRB01, HPR06]. De plus, il s'agit des contraintes pour lesquelles nous proposons, dans la partie Contributions, de nouvelles versions relaxées. Enfin, d'un point de vue applicatif, nous montrons l'apport de ces versions relaxées pour la modélisation et la résolution des « Nurse Rostering Problems » (`NRPs`).

Pour chacune de ces trois contraintes, nous suivrons le même plan de présentation : Définition, Représentation, Test de cohérence et Filtrage.

2.1 Contraintes globales

Une contrainte globale représente généralement un ensemble de propriétés que doivent satisfaire n variables, et qu'il est souvent très agréable de formuler par une seule contrainte plutôt que par une conjonction de contraintes (élémentaires ou pas) qui soit équivalente. Par exemple, la contrainte `AllDifferent`$(\{X_1, \ldots, X_n\})$ permet de représenter en une unique contrainte un ensemble de $n \times (n-1)/2$ contraintes binaires de différence.

L'apport des contraintes globales ne situe pas uniquement d'un point de vue modélisation. En effet, en raisonnant non plus de manière locale, mais de manière globale, des algorithmes de filtrage très efficaces vont pouvoir être développés. Dans tous les cas, une cohérence plus forte que la cohérence d'arc pourra être maintenue (la figure 2.1 présente un problème où l'incohérence ne peut pas être détectée par l'arc cohérence mais peut être facilement détectée en étudiant le problème dans son ensemble). Assez souvent, la cohérence globale peut être maintenue dans des temps tout à fait acceptables. C'est notamment le cas des trois contraintes étudiées dans ce chapitre. Les contraintes globales sont généralement un élément clef pour la résolution de problèmes de grandes tailles [Sim07].

L'efficacité du filtrage des contraintes globales repose bien souvent sur l'utilisation de modèles et algorithmes issus d'autres domaines de l'informatique comme par exemple la programmation linéaire, la théorie des graphes, les flots, etc. L'utilisation de ces modèles et algorithmes, déjà très étudiés par ailleurs, permet l'écriture de tests de cohérence et d'algorithmes de filtrage très efficaces en temps et peu coûteux en espace.

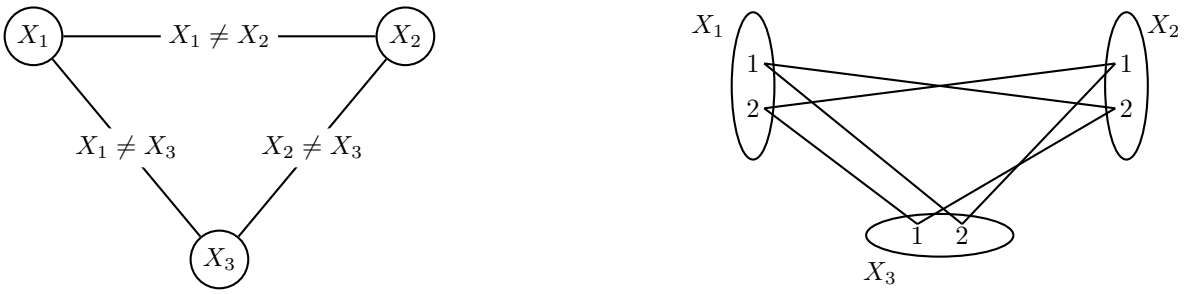


FIG. 2.1 – Exemple de réseau de contraintes où l'incohérence peut être détectée en raisonnant de manière globale et non locale.

Dans la suite de ce mémoire, nous utiliserons les notations suivantes :
- $\mathbf{n} = |\mathcal{X}|$, le nombre de variables impliquées dans la contrainte globale ;
- $\mathbf{Doms} = \bigcup_{X_i \in \mathcal{X}} D_{X_i}$, l'union des domaines ;
- $\mathbf{k} = |Doms|$, la taille de l'union des domaines ;
- $\mathbf{m} = \sum_{X_i \in \mathcal{X}} |D_{X_i}|$, la somme des cardinaux des domaines.

2.2 La contrainte globale AllDifferent($\mathcal{X}$)

2.2.1 Sémantique

La contrainte globale AllDifferent($\mathcal{X}$) est l'une des premières contraintes globales à avoir éré présentée. On retrouve dans ALICE [Lau78] le mot clé DIS qui impose à un ensemble de variables de prendre des valeurs différentes. La contrainte AllDifferent($\mathcal{X}$) permet de représenter en une seule contrainte un ensemble de $n \times (n-1)/2$ contraintes binaires de différence. La contrainte AllDifferent($\mathcal{X}$) est très utile pour modéliser des problèmes d'allocation de ressources.

Définition 39 : AllDifferent($\mathcal{X}$)
Soit l'ensemble de variables $\mathcal{X} = \{X_1, ..., X_n\}$, la contrainte AllDifferent($\mathcal{X}$) *admet une solution* ssi *il existe une instanciation complète telle que les variables de $\mathcal{X}$ soient deux à deux différentes.*

Jean-Charles RÉGIN propose, dans [Rég94], un algorithme permettant d'établir la cohérence globale grâce à la théorie des graphes (des résultats similaires ont été proposés simultanément par Marie-Christine COSTA dans [Cos94]).

Dans la suite de cette section, pour illustrer le principe du test de cohérence et de l'algorithme de filtrage, nous utiliserons l'exemple suivant :

Exemple 3 :
Soit le réseau de contraintes $P=(\mathcal{X}, \mathcal{D}, \mathcal{C})$ avec l'ensemble de variable $\mathcal{X}=\{X_1, X_2, X_3, X_4, X_5\}$ qui ont pour domaines $D_{X_1}=D_{X_2}=\{1,2\}$, $D_{X_3}=\{2,3\}$ et $D_{X_4}=D_{X_5}=\{3,4,5,6\}$ et pour seule

contrainte la contrainte `AllDifferent`($\mathcal{X}$).

2.2.2 Représentation sous forme d'un graphe bipartite

Toute contrainte `AllDifferent`($\mathcal{X}$) peut être représentée par un graphe bipartite $B = (\mathcal{X}, Doms, E)$ où $\mathcal{X}$ est l'ensemble des variables et $Doms$ est l'ensemble des valeurs (l'union des domaines). Deux sommets $X_i \in \mathcal{X}$ et $v_j \in Doms$ sont liés par une arête *ssi* $v_j \in D_{X_i}$. Ce graphe bipartite est aussi appelé *graphe de valeurs* [Rég94] (voir sous-section 1.1.3).

Définition 40 : ***Graphe de valeurs associé à*** `AllDifferent`***($\mathcal{X}$) [Rég94]***
Soit la contrainte `AllDifferent`*($\mathcal{X}$), son graphe de valeurs est défini par $B=(\mathcal{X}, Doms, E)$ avec :*

$$E = \{\{X_i, v_j\} \mid X_i \in \mathcal{X}, v_j \in Doms \text{ t.q. } v_j \in D_{X_i}\}$$

La figure 2.2 représente le graphe de valeurs associé à la contrainte `AllDifferent`($\mathcal{X}$) présentée dans l'exemple 3.

2.2.3 Test de cohérence et filtrage

Il existe une bijection entre les couplages du graphe de valeurs et les instanciations partielles $\mathcal{A}_p$ dont toutes les valeurs sont différentes. L'arête $\{X_i,v_j\}$ appartient à M *ssi* $(X_i=v_j)$ appartient à $\mathcal{A}_p$. Par extension, il existe une bijection entre les couplages couvrant $\mathcal{X}$ et les solutions de la contrainte `AllDifferent`($\mathcal{X}$).

Théorème 1 : ***Test de cohérence de la contrainte*** `AllDifferent`***($\mathcal{X}$)[Rég94]***
Une contrainte `AllDifferent`*($\mathcal{X}$) admet une solution* ssi *il existe un couplage M couvrant $\mathcal{X}$ dans le graphe de valeurs associé.*

La recherche d'un couplage M couvrant $\mathcal{X}$ peut être réalisée grâce à l'algorithme de `Hopcroft & Karp` [HK73] en $O(\sqrt{|\mathcal{X}|} \times |E|)$, c'est-à-dire en $O(\sqrt{n} \times m)$. L'algorithme 2 décrit le test de cohérence de la contrainte `AllDifferent`. La figure 2.3 représente un des couplages couvrants de l'exemple 3 qui correspond à la solution $\{(X_1=1),(X_2=2),(X_3=3),(X_4=4),(X_5=5)\}$.

Algorithme 2 : Test de cohérence de la contrainte `AllDifferent`($\mathcal{X}$).

```
1 fonction ccAllDifferent(X : variables)
2 début
3 |   B ←construireGrapheDeValeurs(X)
4 |   M ←calculerCouplageMaximal(B)
5 |   si |X|=|M| alors retourner true
6 |   sinon retourner false
7 fin
```

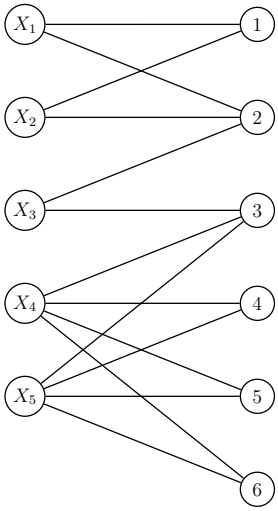


FIG. 2.2 – Graphe de valeurs associé à la contrainte AllDifferent($\mathcal{X}$) de l'exemple 3.

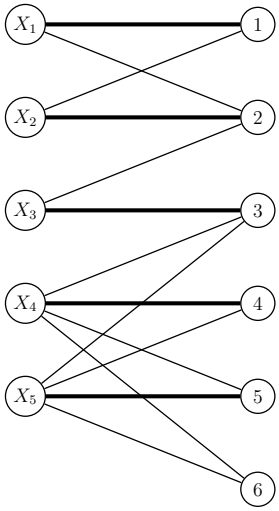


FIG. 2.3 – Les arcs en gras représentent un couplage couvrant $\mathcal{X}$ pour la contrainte AllDifferent($\mathcal{X}$) de l'exemple 3.

Pour établir la cohérence globale, il est nécessaire de filtrer toutes les valeurs qui ne participent pas à au moins une solution, c'est-à-dire à au moins un couplage couvrant $\mathcal{X}$ dans B. Afin d'éviter d'utiliser de manière naïve le test de cohérence pour chaque valeur, il est possible de caractériser toutes les arêtes participant à au moins un couplage couvrant $\mathcal{X}$ grâce à la propriété suivante, issue des travaux de Claude BERGE.

Propriété 3 : ***Appartenance d'une arête à au moins un couplage maximal [Ber70]*** *Une arête appartient à au moins un couplage maximal* ssi *pour un couplage maximal arbitraire M, elle appartient :*

- *soit à M,*
- *soit à une chaîne alternée de longueur paire partant d'un sommet libre,*
- *soit à un cycle alterné de longueur paire.*

Tout couplage couvrant $\mathcal{X}$ étant maximal, il est possible d'utiliser la propriété 3 pour éliminer toutes les arêtes n'appartenant pas à au moins un couplage couvrant.

Pour savoir si une arête du graphe bipartite appartient à un cycle alterné de longueur paire, il suffit d'orienter le graphe de la manière suivante. Si l'arête $\{X_i,v_j\}$ appartient à M alors elle est orientée dans le sens $(X_i \rightarrow v_j)$ sinon $(v_j \rightarrow X_i)$. On notera D_M le graphe orienté de cette façon pour le couplage M. Deux sommets appartiennent à un cycle alterné de longueur paire dans B *ssi* ils appartiennent à la même composante fortement connexe dans D_M.

À partir de la propriété 3 et du théorème 1, on obtient le corollaire 1 permettant de tester la viabilité d'une valeur.

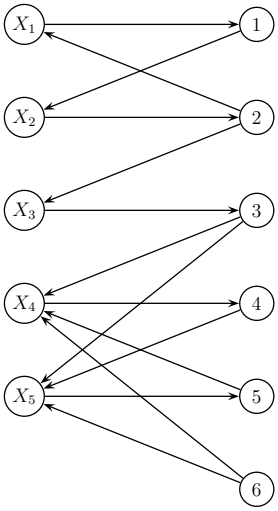


FIG. 2.4 – Orientation du graphe de valeurs de l'exemple 3 par rapport au couplage de la figure 2.3.

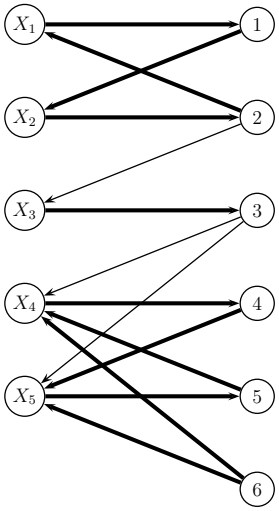


FIG. 2.5 – Les arcs en gras représentent les valeurs viables pour la contrainte `AllDifferent`($\mathcal{X}$) de l'exemple 3.

Corollaire 1 : ***Viabilité d'une valeur pour la contrainte*** `AllDifferent`***($\mathcal{X}$) [Rég94]***
Soit M un couplage couvrant $\mathcal{X}$ dans le graphe de valeurs associé à une contrainte `AllDifferent`*($\mathcal{X}$), une valeur (X_i, v_j) est viable* ssi *elle vérifie l'une des trois conditions suivantes :*

- *$\{X_i, v_j\}$ appartient à M ;*
- *$\{X_i, v_j\}$ appartient à une chaîne alternée de longueur paire issue d'un sommet libre ;*
- *les deux sommets X_i et v_j appartiennent à la même composante fortement connexe dans D_M.*

Dans la figure 2.5, les arêtes en gras représentent les valeurs viables de la contrainte `AllDifferent`($\mathcal{X}$) présentée dans l'exemple 3. Les valeurs $(X_3, 2)$,$(X_4, 3)$ et $(X_5, 3)$ ne sont pas marquées comme viables et doivent être retirées pour rendre la contrainte `AllDifferent`($\mathcal{X}$) globalement cohérente.

La transformation de B en D_M est réalisable en temps linéaire en la taille du graphe bipartite. Le calcul des composantes fortement connexes peut être effectué en $O(n + k + m)$, grâce à l'algorithme de `Tarjan` [Tar72]. L'appartenance à une chaîne alternée paire peut être réalisée grâce au parcours en largeur de D_M en $O(m)$.

Une contrainte `AllDifferent`($\mathcal{X}$) peut être rendue globalement cohérente en $O(n + k + m)$ [Rég94]. L'algorithme 3 permet d'établir la cohérence globale.

2.3 La contrainte globale de cardinalité : `Gcc`$(\mathcal{X}, l, u)$

2.3.1 Sémantique

La contrainte globale de cardinalité (`Gcc`) restreint le nombre d'occurrences de certaines valeurs v_j à respecter une borne inférieure l_j et supérieure u_j.

Algorithme 3 : Maintien la cohérence globale pour AllDifferent($\mathcal{X}$)

```
1  procédure filterAllDifferent(X : variables, B : graphe bipartite, M : couplage maximal)
2  début
3      D_M ← construireGrapheOriente(B, M)
4      CFC ← calculerComposantesFortementConnexes(D_M)
5      P ← ∅
6      pour chaque sommet s libre de D_M faire
7          P ← P ∪ calculerChaineAlternantePaire(s,D_M)
8      pour chaque X_i ∈ X faire
9          pour chaque v_j ∈ D_{X_i} faire
10             si (X_i, v_j) ∉ (M ∪ P) et CFC[X_i] ≠ CFC[v_j] alors
11                 D_{X_i} ← D_{X_i} \ {v_j}
12 fin
```

Définition 41 : Gcc*($\mathcal{X}, l, u$) [Rég96]*

Soit $\mathcal{X} = \{X_1, ..., X_n\}$ un ensemble de variables, soit Doms l'union des domaines de $\mathcal{X}$. À chaque valeur $v_j \in Doms$ est associée une borne inférieure l_j et une borne supérieure u_j. Une contrainte Gcc*($\mathcal{X}$,[$l_1, ..., l_m$],[$u_1, ..., u_m$]) admet une solution* ssi *il existe une instanciation complète de $\mathcal{X}$ telle que :*

$$\forall v_j \in Doms,\ l_j \leq |\{X_i \in \mathcal{X} \mid X_i = v_j\}| \leq u_j$$

La contrainte Gcc($\mathcal{X}, l, u$) est une contrainte très utile notamment pour modéliser des problèmes d'allocation de ressources tels que l'allocation de pièces à des machines. Dans ce type de problèmes, les variables représentent des pièces devant être usinées et les valeurs de leur domaine les machines pouvant effectuer ce traitement. Les bornes inférieures modélisent le nombre minimal de pièces nécessaires pour la mise en route d'une machine et les bornes supérieures la capacité maximale de chaque machine-outil.

Pour illustrer le test de cohérence et le principe de l'algorithme de filtrage, nous utiliserons l'exemple suivant.

Exemple 4 :

Soit $\mathcal{X} = \{X_1, X_2, X_3, X_4, X_5\}$ un ensemble de variables de domaines D_{X_1}=D_{X_2}=D_{X_3}=$\{1, 2\}$, D_{X_4}=$\{2, 3, 4\}$, D_{X_5}=$\{2, 4\}$ et la contrainte Gcc($\mathcal{X}$,[1, 1, 0, 0],[2, 1, 2, 2]).

2.3.2 Représentation sous la forme d'un réseau

Toute contrainte Gcc($\mathcal{X}, l, u$) peut être représentée par un réseau [Rég96]. Au graphe de valeurs de la contrainte Gcc($\mathcal{X}, l, u$) sont ajoutés deux sommets s (la source) et t (le puits). Le sommet s est connecté à tous les sommets représentant des variables et ces arcs ont pour demande et capacité la valeur 1 (une variable doit prendre une et une seule valeur). Les arcs du graphe de valeurs sont orientés des variables vers les valeurs et ont pour demande 0 et

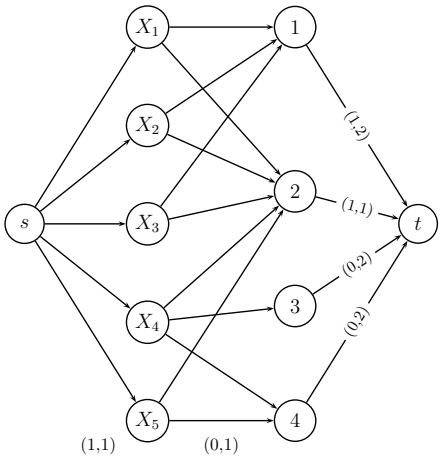


FIG. 2.6 – Réseau associé à la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ de l'exemple 4.

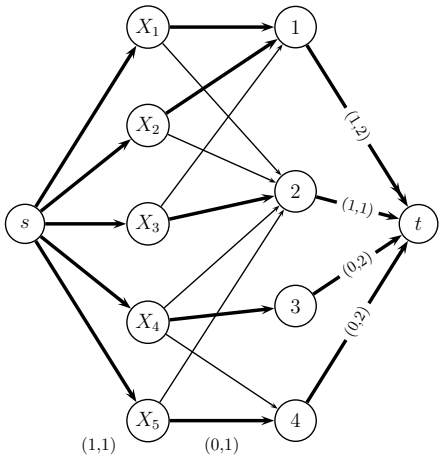


FIG. 2.7 – Les arcs en gras représentent un flot faisable de valeur 5.

pour capacité 1 (une variable doit prendre une valeur parmi celles de son domaine). Chaque sommet représentant une valeur v_j est connecté au puits par un arc de demande égale à la borne inférieure l_j et de capacité égale à la borne supérieure u_j.

Définition 42 : ***Réseau associé à une contrainte*** $\texttt{Gcc}(\mathcal{X}, l, u)$
La contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ *peut être représentée par le réseau* $R=(V, A)$ *avec :*

$$V = \{s\} \cup \mathcal{X} \cup Doms \cup \{t\} \quad avec \quad A_s = \{(s \rightarrow X_i) \mid X_i \in \mathcal{X}\}$$
$$A = A_s \cup A_{\mathcal{X}} \cup A_t \qquad\qquad A_{\mathcal{X}} = \{(X_i \rightarrow v_j) \mid X_i \in \mathcal{X}, v_j \in D_{X_i}\}$$
$$A_t = \{(v_j \rightarrow t) \mid v_j \in Doms\}$$

avec pour demandes et capacités :

$$si\ a \in A_s, \qquad d(a) = 1\ et\ c(a) = 1$$
$$si\ a \in A_{\mathcal{X}}, \qquad d(a) = 0\ et\ c(a) = 1$$
$$si\ a = (v_j \rightarrow t) \in A_t, \qquad d(a) = l_j\ et\ c(a) = u_j$$

La figure 2.6 présente le réseau associé à l'exemple 4.

2.3.3 Test de cohérence et filtrage

Par construction du réseau, il existe une bijection entre les flots faisables de valeur n dans le réseau R (voir section 1.5) et les solutions de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$. Chaque arc $(X_i \rightarrow v_j)$ alimenté par une unité de flot correspond à l'affectation d'une valeur v_j à une variable X_i.

Théorème 2 : ***Test de cohérence de*** $\texttt{Gcc}(\mathcal{X}, l, u)$ ***[Rég96]***
Une contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ *admet une solution* ssi *il existe un flot faisable de valeur* n *dans le réseau associé.*

L'algorithme 4 permet de tester la cohérence d'une contrainte Gcc. Le calcul d'un tel flot faisable dans un réseau est réalisable en O($n \times SP$), avec SP le temps nécessaire au calcul d'un s–t chemin, grâce à l'algorithme de Ford & Fulkerson [FF56]. Soit un temps total en $O(n \times m)$. La figure 2.7 décrit un flot faisable associé à la solution {(X_1=1),(X_2=1),(X_3=2), (X_4=3),(X_5=4)}.

Algorithme 4 : Test de cohérence de la contrainte Gcc($\mathcal{X}, l, u$).

```
1 fonction ccGcc(X : variables, l : bornes inférieures, u : bornes supérieures)
2 début
3 |   R ← construireReseau(X, l, u)
4 |   f ← calculerFlot(R)
5 |   si f est faisable alors retourner true
6 |   sinon retourner false
7 fin
```

Une valeur (X_i, v_j) est viable *ssi* l'arc $(X_i \rightarrow v_j)$ appartient à au moins un flot faisable de valeur n. Pour rendre la contrainte globalement cohérente et afin d'éviter le calcul d'un tel flot pour chaque valeur, on utilise la propriété suivante permettant de savoir si un arc appartient à au moins un flot faisable de valeur n à partir d'un flot faisable initial.

Propriété 4 : ***Appartenance d'un arc à au moins un flot faisable [Rég96]***
Un arc (u→v) participe à un moins un flot faisable ssi *pour un flot faisable arbitraire f, une des conditions suivantes est remplie :*

- *l'arc (u→v) participe à f,*
- *les sommets u et v appartiennent à la même composante fortement connexe dans le réseau résiduel engendré par f.*

Ainsi grâce au théorème 2 et à la propriété 4, on obtient le corollaire 2 permettant de tester la viabilité d'une valeur (X_i, v_j).

Corollaire 2 : ***Viabilité d'une valeur*** (X_i, v_j) ***pour une contrainte*** Gcc($\mathcal{X}, l, u$) ***[Rég96]***
Soit une contrainte Gcc($\mathcal{X}, l, u$)*, R le réseau associé et f un flot faisable de valeur n dans R. Une valeur (X_i, v_j) est viable* ssi *elle satisfait une des deux conditions suivantes :*

- *l'arc (X_i→v_j) est utilisé par le flot f.*
- *les sommets X_i et v_j appartiennent à la même composante fortement connexe dans le réseau résiduel de f.*

À partir d'un flot initial faisable f de valeur n (calculé durant le test de cohérence), il suffit, pour rendre la contrainte globalement cohérente, de calculer les composantes fortement connexes dans le réseau résiduel engendré par f (voir la section 1.5 pour plus de détails). Ce calcul peut être réalisé en $O(n + k + m)$ grâce à l'algorithme de Tarjan [Tar72].

Comme le montre la figure 2.8, les valeurs $(X_4, 2)$ et $(X_5, 2)$ ne sont pas viables. La contrainte

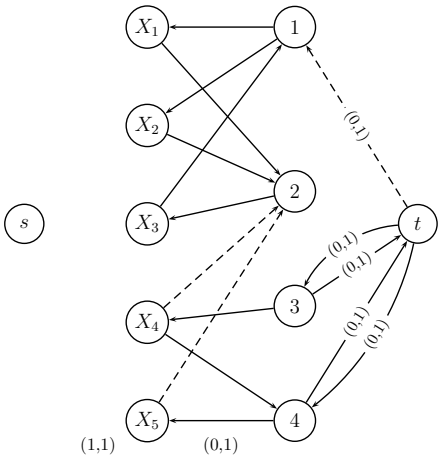


FIG. 2.8 – Réseau résiduel du flot présenté dans la figure 2.7. Les arcs en pointillés représentent des arcs n'appartenant pas au flot initial et dont les extrémités n'appartiennent pas à la même composante fortement connexe.

Algorithme 5 : Maintien de la cohérence globale pour $\texttt{Gcc}(\mathcal{X}, l, u)$.[Rég96]

```
1 procédure filter-gcc(X : variables, l et u : bornes, R : réseau , f : flot faisable)
2 début
3     Soit R_f le réseau résiduel de R engendré par f
4     CFC ← calculerComposantesFortementConnexes(R_f)
5     pour chaque X_i ∈ X faire
6         pour chaque v_j ∈ D_{X_i} faire
7             si (X_i → v_j) ∉ f et CFC[X_i] ≠ CFC[v_j] alors
8                 D_{X_i} ← D_{X_i} \ {v_j}
9 fin
```

$\texttt{Gcc}(\mathcal{X}, l, u)$ de l'exemple 4 peut être rendue globalement cohérente en retirant ces deux valeurs. L'algorithme 5 décrit le maintien de la cohérence globale pour la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$.

Remarque 4 : Une amélioration de la complexité du test de cohérence a été proposée grâce à la recherche d'un couplage dans le graphe de valeurs de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ [QLOBG04]. Ainsi, le test de cohérence peut être effectué en $O(\sqrt{n} \times m)$ dans le pire cas.

2.3.4 $\texttt{AllDifferent}(\mathcal{X})$ comme un cas particulier de $\texttt{Gcc}(\mathcal{X}, l, u)$

La contrainte $\texttt{AllDifferent}(\mathcal{X})$ est un cas particulier de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$. En effet, imposer que toutes les variables prennent des valeurs différentes revient à imposer que chaque valeur ne peut être utilisée plus d'une fois. Ainsi pour exprimer une contrainte $\texttt{AllDifferent}(\mathcal{X})$ à l'aide d'une contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$, il suffit de fixer toutes les bornes inférieures à 0 et toutes les bornes supérieures à 1.

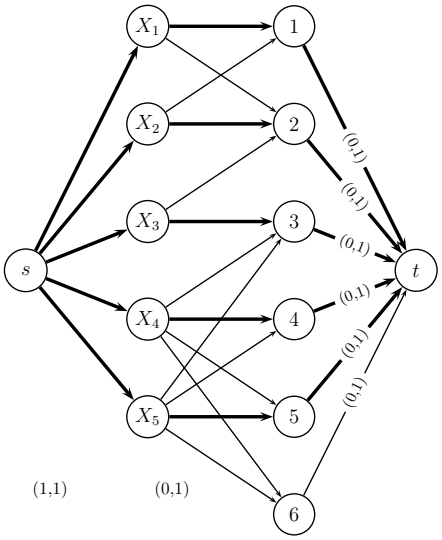


FIG. 2.9 – Solution pour la contrainte `All-Different`($\mathcal{X}$) de l'exemple 3.

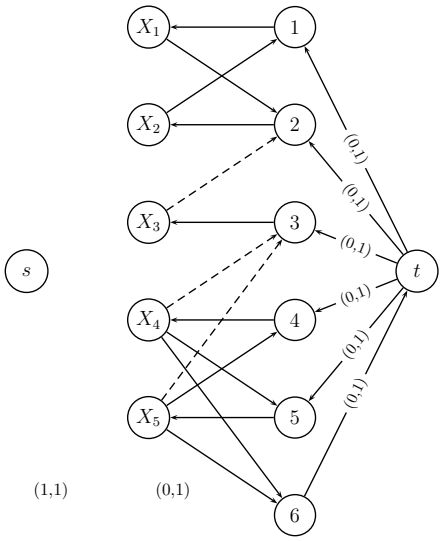


FIG. 2.10 – Réseau résiduel associé à la solution de la figure 2.9.

On peut ainsi reformuler la contrainte `AllDifferent`($\mathcal{X}$) de l'exemple 3. L'ensemble de variables et leurs domaines restent inchangés et la contrainte `AllDifferent`($\mathcal{X}$) s'exprime par la contrainte `Gcc`($\mathcal{X}$,[0,0,0,0,0,0],[1,1,1,1,1,1]).

Définition 43 : ***Réseau associé à une contrainte*** `AllDifferent`***(*** $\mathcal{X}$***)***
Une contrainte `AllDifferent`*(*$\mathcal{X}$*) peut être représentée par le réseau R=(V, A) avec :*

$$V = \{s\} \cup \mathcal{X} \cup Doms \cup \{t\} \quad avec \quad A_s = \{(s \rightarrow X_i) \mid X_i \in \mathcal{X}\}$$
$$A = A_s \cup A_{\mathcal{X}} \cup A_t \qquad\qquad A_{\mathcal{X}} = \{(X_i \rightarrow v_j) \mid X_i \in \mathcal{X}, v_j \in D_{X_i}\}$$
$$A_t = \{(v_j \rightarrow t) \mid v_j \in Doms\}$$

avec pour demandes et capacités :

$$si\ a \in A_s, \quad d(a) = 1\ et\ c(a) = 1$$
$$si\ a \in A_{\mathcal{X}}, \quad d(a) = 0\ et\ c(a) = 1$$
$$si\ a \in A_t, \quad d(a) = 0\ et\ c(a) = 1$$

Les caractérisations de la cohérence de `Gcc`($\mathcal{X}, l, u$) et de la viabilité d'une valeur pour cette contrainte peuvent ainsi être immédiatement réutilisées pour la contrainte `AllDifferent`($\mathcal{X}$) ainsi représentée.

Les figures 2.9 et 2.10 présentent respectivement une solution et le réseau résiduel associé à cette solution. Les arcs en pointillés représentent les valeurs non viables.

La complexité du test de cohérence utilisant cette représentation est en $O(n \times m)$ et celle de l'algorithme de filtrage en $O(n + k + m)$.

2.4 La contrainte globale Regular($\mathcal{X}, \Pi$)

2.4.1 Sémantique

La contrainte globale Regular($\mathcal{X}, \Pi$) impose qu'une séquence de n variables forme un mot reconnu par un automate fini déterministe (voir section 1.6).

Définition 44 : Regular*($\mathcal{X}, \Pi$) [Pes04]*
Soit Π=($Q, \Sigma, \delta, q_0, F$) un automate fini déterministe et $\mathcal{X} = \{X_1, ..., X_n\}$ un ensemble de variables de domaine $D_{X_1}, ..., D_{X_n} \subseteq \Sigma$, la contrainte Regular*($\mathcal{X}, \Pi$) impose que la séquence des valeurs de $\mathcal{X}$ appartient au langage régulier reconnu par Π.*

Cette contrainte permet notamment de modéliser, de manière très élégante, les règles de séquence d'un problème d'affectation de personnels. Les variables impliquées dans la contrainte Regular($\mathcal{X}, \Pi$) représentent les créneaux durant lesquels un employé doit être affecté à une équipe de travail. Les valeurs du domaine d'une variable représentent les différentes équipes affectables. L'automate permet alors de modéliser les règles de transition entre ces activités (par exemple, après avoir travaillé trois nuits consécutives, un employé doit être de repos au moins deux jours de suite).

2.4.2 Représentation sous forme d'un graphe en couches

Une contrainte Regular($\mathcal{X}, \Pi$) peut être représentée sous forme d'un graphe en couches [Pes04]. Dans le graphe en couches associé à une contrainte Regular($\mathcal{X}, \Pi$), chaque couche représente une variable et chacun des sommets de cette couche correspond à un état de l'automate Π. Le sommet q_l^i représente l'état q_l associé à la i-ième couche. Les arcs représentent les transitions de l'automate. Le sommet s est connecté au sommet de la première couche représentant l'état initial q_0 de Π. Tous les sommets de la dernière couche représentant un état final $q \in F$ sont connectés à t.

Définition 45 : ***Graphe en couches associé à une contrainte*** Regular*(X, Π) [HPR06]*
Le graphe en couches associé à une contrainte Regular*(X, Π) est un graphe orienté $G = (V, A)$ tel que :*

$$V = \{s\} \cup V_0 \cup ... \cup V_n \cup \{t\}$$
$$A = A_s \cup A_1 \cup ... \cup A_n \cup A_t$$
avec
$$\forall i \in [0..n],\ V_i = \{q_l^i \mid q_l \in Q\}$$
$$A_s = \{(s \rightarrow q_0^0)\}$$
$$\forall i \in [1..n],\ A_i = \{(q_l^{i-1} \rightarrow q_m^i, v_j) \mid v_j \in D_{X_i} \text{ et } \delta(q_l, v_j) = q_m\}$$
$$A_t = \{(q_l^n \rightarrow t) \mid q_l \in F\}$$

Pour illustrer la contrainte Regular($\mathcal{X}, \Pi$), nous utiliserons l'exemple 5.

Exemple 5 :
Soit l'automate Π de la figure 2.11, l'ensemble de variables $\mathcal{X}$=$\{X_1, X_2, X_3, X_4, X_5\}$ avec les domaines D_{X_1}=D_{X_2}=D_{X_3}=D_{X_4}=D_{X_5}=$\{a, b, c\}$ et la contrainte `Regular`($\mathcal{X}$,Π).

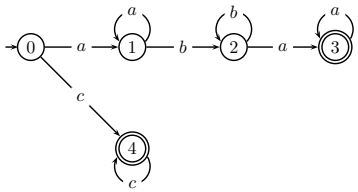


FIG. 2.11 – Automate Π associé à l'exemple 5.

La figure 2.12 présente le graphe en couches associé à la contrainte `Regular`$(\mathcal{X}, \Pi)$ de l'exemple 5.

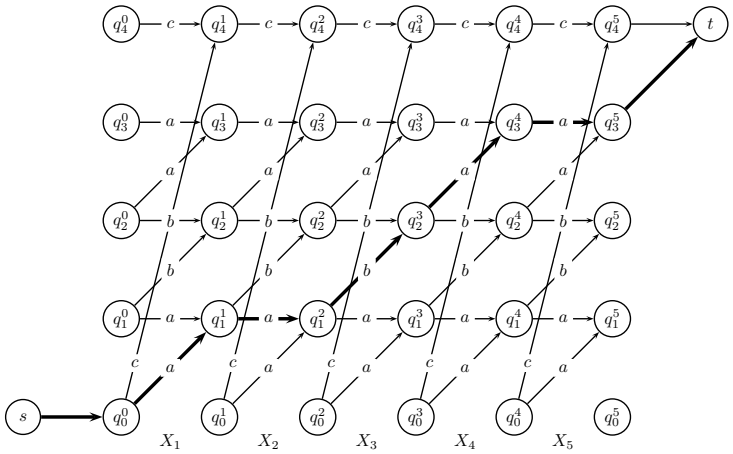


FIG. 2.12 – Graphe en couches associé à la contrainte `Regular`$(\mathcal{X}, \Pi)$ de l'exemple 5.

2.4.3 Test de cohérence et filtrage

Par construction du graphe en couches, la séquence d'étiquettes le long d'un s–t chemin représente un mot reconnu par l'automate Π, et correspond donc à une solution pour la contrainte globale `Regular`$(\mathcal{X}, \Pi)$.

Théorème 3 : ***Test de cohérence de*** `Regular`***$(\mathcal{X}, \Pi)$ [Pes04]***
Une contrainte `Regular`$(\mathcal{X}, \Pi)$ *admet une solution* ssi *il existe au moins un s–t chemin dans le graphe en couches associé.*

Comme le montre le graphe en couches de la figure 2.12, $\{s, q_0^0, a, q_1^1, a, q_1^2, b, q_2^3, a, q_3^4, a, q_3^5, t\}$ est un s–t chemin. Celui-ci représente le mot "aabaa" reconnu par l'automate Π. L'instanciation complète $\{(X_1 = a), (X_2 = a), (X_3 = b), (X_4 = a), (X_5 = a)\}$ est bien une solution pour la contrainte `Regular`($\mathcal{X}$,Π) de l'exemple 5.

Algorithme 6 : Construction du graphe en couches réduit pour `Regular`$(\mathcal{X}, \Pi)$. [Pes04]

```
1  fonction construireGrapheCouchesReduit(X : variables, Π : automate)
2  début
3      /* phase ascendante */
4      ajouter arc (s → q_0^0,void)
5      Soit Fi une file vide
6      Fi ← Fi.enfiler(q_0^0)
7      tant que Fi est non vide faire
8          q_l^i ← Fi.defiler()
9          pour chaque δ(q_l, v_j) = q_m t.q. v_j ∈ D_{X_{i+1}} faire
10             ajouter arc (q_l^i → q_m^{i+1}, v_j)
11             si q_m^{i+1} ∉ Fi alors Fi ← Fi.empiler(q_m^{i+1})
12     pour chaque q_l ∈ F et q_l^n possède au moins un arc entrant faire
13         ajouter arc (q_l^n → t,void)
14     /* phase descendante */
15     pour chaque i ∈ {n, .., 1} faire
16         pour chaque m ∈ [0, .., |Q| −1] faire
17             si q_m^i n'as pas d'arc sortant alors
18                 retirer tous les arcs entrant de q_m^i
19     retourner le graphe en couches
20 fin
```

Le graphe en couches étant par construction acyclique, il est possible de tester la cohérence de la contrainte `Regular`$(\mathcal{X}, \Pi)$ grâce à un parcours en largeur du graphe en $O(n \times |Q| \times |\Sigma|)$.

Le théorème précédent peut être utilisé pour tester la viabilité d'une valeur. En effet, une valeur (X_i, v_j) est viable *ssi* il existe au moins un s–t chemin passant par un arc reliant les couches $(i-1)$ et i étiqueté par la valeur v_j. Ainsi, pour établir la cohérence globale on pourrait calculer $(n \times |Q| \times |\Sigma|)$ chemins au sein du graphe en couches. Pour éviter cela, il est nécessaire de construire un graphe en couches réduit, dans lequel seuls les arcs appartenant à un s–t chemin sont conservés [Pes04].

Cette méthode procède en deux phases :

- la première phase, dite ascendante, duplique les transitions comme précédemment mais seuls les arcs accessibles à partir de s sont construits ;
- la seconde phase, dite descendante, élimine tous les arcs non connectés à t en parcourant de droite à gauche le graphe en couches.

L'algorithme 6 présente la construction du graphe en couches réduit grâce à cette méthode en deux phases. Cette construction peut être réalisée en $O(n \times |Q| \times |\Sigma|)$.

La figure 2.13 représente le graphe en couches réduit construit par la phase ascendante pour l'exemple 5. Le graphe en couches réduit de la figure 2.14 est le résultat de l'application de la seconde phase sur ce même exemple.

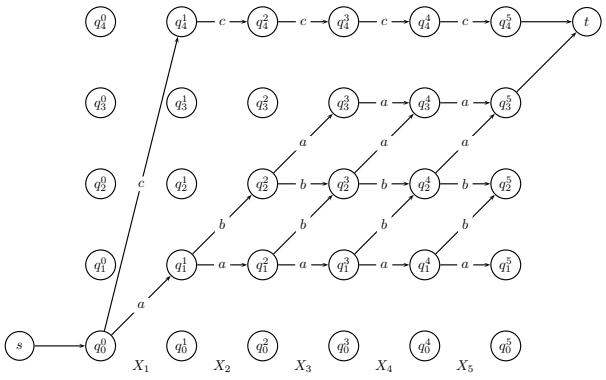


FIG. 2.13 – Graphe en couches réduit associé à l'exemple 5 après la phase ascendante.

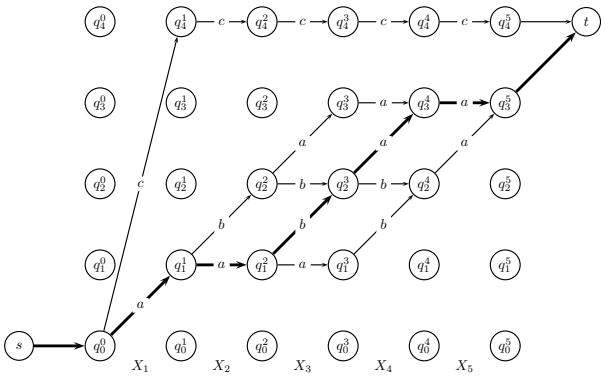


FIG. 2.14 – Graphe en couches réduit associé à l'exemple 5 après la phase descendante.

À l'issue de la génération du graphe en couches réduit, seuls des arcs appartenant à au moins un s–t chemin sont présents. Ainsi, pour tester la cohérence il suffit de vérifier l'existence d'au moins un arc dans A_1 (voir l'algorithme 7 et la définition 45).

Algorithme 7 : Test de cohérence de la contrainte Regular$(\mathcal{X}, \Pi)$.

```
1 fonction ccRegular(X : variables, Π : automates)
2 début
3 |   G ← construireGrapheCouchesReduit(X, Π)
4 |   si |A1| ≠ ∅ alors retourner true
5 |   sinon retourner false
6 fin
```

Pour filtrer les valeurs incohérentes, il suffit de parcourir les différentes couches du graphe, et d'éliminer les valeurs n'étant utilisées par aucun arc.

Corollaire 3 : ***Viabilité d'une valeur*** (X_i, v_j) ***pour*** Regular$(\mathcal{X}, \Pi)$ ***[Pes04]***
Soit G le graphe en couches réduit (construit selon la méthode en deux phases) associé à la contrainte Regular$(\mathcal{X}, \Pi)$*, une valeur (X_i, v_j) est viable* ssi *il existe au moins un arc reliant les couches $(i-1)$ et i étiqueté par la valeur v_j dans G.*

Ainsi, il est possible de déterminer si une valeur (X_i, v_j) est viable en parcourant les arcs de la couche $(i-1)$. La cohérence globale peut ainsi être établi en parcourant le graphe en couches, c'est-à-dire en $O(n \times |Q| \times |\Sigma|)$ dans le pire des cas (voir algorithme 8).

Dans le cas de l'exemple 5, aucun arc étiqueté par b ne relie les sommets de V_0 à ceux de V_1. Par conséquent, la valeur b peut être retirée du domaine de X_1. La contrainte Regular$(\mathcal{X}, \Pi)$ de l'exemple 5 peut être rendue globalement cohérente en filtrant les valeurs (X_1, b) et (X_5, b).

Algorithme 8 : Algorithme de filtrage de `Regular`$(\mathcal{X}, \Pi)$.

```
1 procédure filterRegular(𝒳 : variables, G : graphe en couches réduit)
2 début
3     pour chaque X_i ∈ 𝒳 faire
4         pour chaque v_j ∈ D_{X_i} faire
5             si ∄ arc étiqueté par v_j ∈ A_i alors D_{X_i} → D_{X_i}\{v_j}
6 fin
```

2.5 Conclusion

Dans ce chapitre, nous avons étudié trois contraintes globales : `AllDifferent`, `Gcc` et `Regular`. Pour chacune de ces contraintes, nous avons caractérisé le test de cohérence et présenté un algorithme de filtrage permettant de maintenir la cohérence globale en temps polynomial (voir Tableau 2.1).

Contrainte	Modèle	Cohérence	Filtrage	Référence
`AllDifferent`	Graphe bipartite	$O(\sqrt{n} \times m)$	$O(m)$	[Rég94]
	Réseau	$O(n \times m)$	$O(m)$	[Rég96]
`Gcc`	Réseau	$O(n \times m)$	$O(m)$	[Rég96]
	Graphe bipartite	$O(\sqrt{n} \times m)$	$O(m)$	[QLOBG04]
`Regular`	Graphe en couches	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$	[Pes04]

TAB. 2.1 – Tableau récapitulatif des mises en œuvre.

Toutefois, ce cadre ne permet pas de modéliser des situations où les problèmes n'admettent pas de solution, ce qui est très souvent le cas dans les applications réelles. La relaxation de contraintes, et plus particulièrement de contraintes globales, fera l'objet des deux prochains chapitres.

Chapitre 3

Relaxation de contraintes globales

Sommaire

De nombreux problèmes réels sont par nature sur-contraints (ils ne possèdent aucune solution). Cela est souvent dû à un ensemble d'exigences trop fortes des spécifieurs ou des utilisateurs de l'application, ce qui conduit à l'introduction de contraintes antagonistes. Ainsi, dans les problèmes d'emplois du temps, les souhaits ou préférences exprimés pas les élèves, les enseignants et l'administration sont souvent incompatibles.

Différents modèles ou cadres pour la relaxation des CSPs ont été proposés : les CSPs hiérarchiques [BFBW92], les Partial CSPs [FW92], les CSPs valués [SFV95], les Semi-ring CSPs [BMR+99], la relaxation disjonctive du réseau de contraintes [Pet02], etc.

L'objectif de ce chapitre est de motiver notre choix du modèle de la relaxation disjonctive [Pet02] pour la relaxation des contraintes globales. Pour cela, nous définissons la relaxation d'une contrainte (quelconque) et les propriétés inhérentes que doit vérifier la relaxation pour pouvoir modéliser et résoudre des problèmes sur-contraints. Puis, nous passons brièvement en revue les différents cadres proposés en discutant leurs avantages et leurs inconvénients au regard de ces propriétés. Nous motivons alors notre choix du modèle de la relaxation disjonctive pour la relaxation des contraintes globales et nous présentons ce modèle plus avant en définissant la notion de sémantique de violation, de cohérence et de filtrage associés.

3.1 Relaxer une contrainte

Relaxer une contrainte (quelconque) c'est autoriser celle-ci à ne pas être nécessairement satisfaite contre un coût. Dans ce mémoire, nous adopterons la convention suivante : plus le coût associé sera grand, plus la contrainte sera jugée importante. Le coût d'une contrainte doit permettre la formulation de plusieurs propriétés essentielles comme l'expression de différents niveaux de priorité ou de mesures de violation fines. Il est aussi essentiel de pouvoir modéliser un ensemble de règles régissant la violation.

3.1.1 Contraintes d'intégrité et contraintes de préférence

Lorsque l'on cherche à modéliser un problème sur-contraint, on distingue deux catégories de contraintes :

1. Les *contraintes d'intégrité* (ou dures – « hard » en anglais) qui doivent être impérativement satisfaites. Les contraintes d'intégrité reflètent généralement des obligations d'ordre physique comme le fait qu'une infirmière ne peut pas travailler dans deux services hospitaliers de manière simultanée.
2. Les *contraintes de préférence* (ou molles – « soft » en anglais) qui expriment des souhaits formulés sous forme de propriétés que l'on aimerait voir vérifiées par une solution. Ces contraintes de préférence correspondent à des exigences qu'il est souhaitable de respecter pour obtenir une solution de bonne qualité. Pour l'exemple de la création d'emplois du temps pour des infirmières (`NRP`s), il peut s'agir d'un jour de repos demandé par une

infirmière qu'il est préférable de satisfaire (mais qui ne nuira pas au bon fonctionnement du service si jamais ce repos n'est pas accordé).

Le but de la relaxation n'est plus de satisfaire toutes les contraintes (cela n'est pas toujours possible) mais de les satisfaire au mieux. « Au mieux » signifie ici i) satisfaire toutes les contraintes dures et ii) minimiser une agrégation des coûts des contraintes de préférence insatisfaites.

3.1.2 Différents niveaux de priorité

Les contraintes de préférence n'ont pas toute la même importance. Dans l'exemple des NRPs, une contrainte de préférence gérant un repos souhaité par une infirmière peut-elle primer sur une contrainte imposant un nombre minimal d'heures que doit réaliser cette même infirmière ?

3.1.3 Quantifier l'insatisfaction de manière fine

En plus de pouvoir exprimer les coûts des différentes contraintes de préférence en fonction de leur degré d'importance, il est essentiel de pouvoir exprimer le coût de l'insatisfaction en fonction de critères plus fin que la satisfaction de la contrainte ou son insatisfaction. Le coût doit être exprimé de manière plus fine que par une constante.

Le premier critère que doit tenir compte la valuation est le *niveau d'insatisfaction*. Par exemple, un planning où une infirmière doit travailler un jour supplémentaire est préférable à un planning dans lequel celle-ci travaille plusieurs jours supplémentaires.

Un second critère que doit refléter la valuation est la *nature de la violation*. Toujours sur l'exemple d'une infirmière ayant des jours excédentaires à son planning, si ces jours correspondent à des équipes de jour il est raisonnable que la violation soit plus faible que si ces journées travaillées se déroulent en équipe de nuit.

3.1.4 Pouvoir contrôler l'insatisfaction

Le contrôle de l'insatisfaction est un point important pour la découverte de solutions de bonne qualité et exploitables. Un des enjeux à remplir pour le développement d'applications réelles est de pouvoir *répartir l'insatisfaction* selon un ensemble de critères. Il n'est pas raisonnable de penser que, malgré un coût faible, une solution dont toute l'insatisfaction serait reportée sur une seule infirmière soit acceptable. Un autre enjeu important est l'*activation de contraintes* en cas d'évènements comme l'insatisfaction d'une contrainte donnée. En effet, si une infirmière doit commencer son service en avance alors celle-ci ne doit pas terminer en retard le même jour.

3.2 Différents cadres pour la relaxation de contraintes

Dans cette section, nous présentons brièvement les trois principaux cadres proposés en discutant leurs avantages et leurs inconvénients au regard des caractéristiques/propriétés de la relaxation de contraintes présentées dans la section précédente.

– les CSPs hiérarchiques,
– les CSPs valués,
– la relaxation disjonctive de CSPs.

3.2.1 Les CSPs hiérarchiques

Un CSP hiérarchique est défini par un tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C}_H, e, better)$ où $\mathcal{X}$ est un ensemble de variables de domaines $\mathcal{D}$, $\mathcal{C}_H$ est une hiérarchie de contraintes représentée par un vecteur de la forme $(\mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_n)$ où chaque $\mathcal{C}_i$ est un sous-ensemble de contraintes. Les hiérarchies forment une partition de l'ensemble des contraintes, e une fonction d'erreur et *better* un comparateur de solutions.

Pour une hiérarchie de contraintes $\mathcal{C}_H$ donnée, les contraintes d'intégrité constituent le niveau $\mathcal{C}_0$. Les autres niveaux représentent les contraintes qui doivent être préférentiellement satisfaites, du niveau le plus fort $\mathcal{C}_1$ au plus faible $\mathcal{C}_n$.

Une solution est une affectation satisfaisant $\mathcal{C}_0$ et dont la qualité dépend des contraintes des autres niveaux. Une solution est optimale s'il n'existe pas de solution de meilleure qualité au sens du comparateur *better*.

Afin de déterminer le coût de violation d'une instanciation, la fonction d'erreur e associe à chaque contrainte c une valeur réelle positive reflétant le degré de violation de c. Le comparateur *better* définit un ordre partiel entre deux solutions en examinant les violations de chacune des hiérarchies $\mathcal{C}_i$ selon l'ordre croissant de celles-ci. Pour une hiérarchie de contraintes $\mathcal{C}_i$ et deux solutions s_1 et s_2 :

– si s_1 insatisfait plus faiblement $\mathcal{C}_i$ que s_2 alors *better* préfèrera la solution s_1 ;
– si s_1 et s_2 insatisfont de la même façon $\mathcal{C}_i$ alors *better* départagera s_1 et s_2 grâce aux contraintes de la hiérarchie $\mathcal{C}_{i+1}$.

Le lecteur intéressé pourra se référer aux articles de Alan Borning *et al.* [BFBW92] et de Martin Henz *et al.* [HFC+00].

Faiblesses des CSPs hiérarchiques

Premièrement, les problèmes réels sont très rarement exprimés de manière hiérarchique. De plus, à partir de la description d'un problème il est souvent difficile d'extraire une hiérarchisation des contraintes sans ajouter une structuration qui n'existe pas dans l'application d'origine.

Deuxièmement, le comparateur de solutions *better* manque de précision. En effet, lorsque qu'une contrainte d'un niveau $\mathcal{C}_i$ est insatisfaite, alors peu importe que les contraintes des niveaux $\mathcal{C}_j$, pour $i < j \leq n$, soient satisfaites ou pas (ce phénomène est connu sous le nom d'"effet de noyade").

Ainsi, pour les NRPs, il est difficile de hiérarchiser les contraintes. En effet, les contraintes sur les enchaînements d'équipes de travail doivent-elles primer sur les contraintes limitant le nombre d'heures supplémentaires ou pas ? De plus dans des applications comme celle-ci un

Classe	$e \in E$	$\oplus$	$\bot$	$\top$	$\succ$
CSP	$\{true, false\}$	$\wedge$	$true$	$false$	$true \succ false$
Max CSP	{0,1}	$+$	0	$+\infty$	$>$
Weighted CSP	$\overline{\mathbb{N}}$	$+$	0	$+\infty$	$>$
CSP possibilistes	[0,1]	max	0	1	$>$
CSP flous	[0,1]	min	1	0	$<$
CSP lexicographiques	$[0,1]^* \cup \top$	$\cup$	$\emptyset$	$\top$	lexicographique

TAB. 3.1 – Les quatre classes de VCSPs et les cas particuliers des CSPs et des Max CSPs

trop grand nombre de violations de contraintes de faible préférence peuvent rendre un planning inexploitable.

3.2.2 Les CSPs valués (et les Semi-ring CSPs)

Les CSPs valués (VCSPs) et le Semi-ring CSPs sont basés sur le même principe ; à chaque contrainte est associée une valuation dépendante des valeurs affectées aux variables de sa portée. Les valeurs de la valuation sont prises dans un ensemble ordonné E. Si cet ensemble E est totalement ordonné alors il est possible de passer d'un VCSP à une Semi-ring CSP, et vice-versa (voir [BMR+99]). Dans ce qui suit, nous présentons seulement les VCSPs.

Définition 46 : ***Structure de valuation [SFV95]***
Une structure de valuation $\mathcal{S}$ est un triplet $(E, \oplus, \succ)$ tel que :

- *E est un ensemble dont les éléments sont appelés* valuations. *E est totalement ordonné par $\succ$, avec pour élément maximum $\top$ et pour élément minimum $\bot$;*
- *E est muni d'une loi de composition interne $\oplus$, commutative et associative qui vérifie :*
  - *un élément neutre : $\forall a \in E,\ a \oplus \bot = a$*
  - *un élément absorbant : $\forall a \in E,\ a \oplus \top = \top$*
  - *la monotonie : $\forall a, b, c \in E,\ (a \succcurlyeq b) \Rightarrow ((a \oplus c) \succcurlyeq (b \oplus c))$*

Définition 47 : ***CSP valué [SFV95]***
Un CSP valué est un quintuplet $(\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{S}, \varphi)$ où :

- *$(\mathcal{X}, \mathcal{D}, \mathcal{C})$ est un réseau de contraintes,*
- *$\mathcal{S}$ est une structure de valuation,*
- *φ est une application de $\mathcal{C}$ dans E ; $\varphi(c)$ est appelée valuation de c.*

Les différentes classes de CSP valués.

Le tableau 3.1 présente les différentes classes de VCSPs et leur structure de valuation.

CSP classique : Dans le formalisme des CSPs classiques ou $\wedge$-CSPs, une affectation est considérée inacceptable dès qu'une contrainte est insatisfaite. Les tuples interdits (resp. autorisés) ont donc pour valuation $\top$ (resp. $\bot$). L'opérateur d'agrégation est $\wedge$.

Max CSP : Dans [FW92] est présenté le cadre des problèmes de satisfaction maximale de contraintes. Dans ce formalisme, toutes les contraintes de $\mathcal{C}$ ont un même poids w (généralement $w = 1$), lorsqu'une contrainte c est satisfaite $\varphi(c)$=0 et sinon $\varphi(c)$=1. Le problème consiste à minimiser le nombre de contraintes insatisfaites.

Weighted CSP : Pour pouvoir exprimer des préférences entre plusieurs contraintes, Eugene C. Freuder *et al.* proposent une extension des Max CSPs en attribuant à chaque contrainte c un poids exprimant le coût de son insatisfaction (ce poids est appelé « pénalité » dans [SFV97]) [FW92]. L'objectif du problème est de minimiser la somme des poids des contraintes insatisfaites.

CSP possibiliste : Dans ce problème, à chaque contrainte est associée une « priorité » modélisée par un poids compris entre 0 et 1 (voir [Sch92]). Le but de ce problème est de minimiser la priorité maximale des contraintes qui sont insatisfaites.

CSP flou : À chaque tuple est associé une valuation comprise entre 0 et 1. La valuation 1 exprime la priorité la plus faible alors que la valuation 0 dénote la priorité la plus importante [DFP93]. Dans les CSP flous, le coût d'une contrainte n'est pas nécessairement constant, il dépend des valeurs affectées aux variables de la portée de cette contrainte. À l'inverse des CSPs possibilistes, le but est de maximiser la priorité de la contrainte la plus importante.

CSP lexicographique : Dans le formalisme des CSPs lexicographiques [FLS93], l'opérateur d'agrégation est l'union sur multi-ensemble. Chaque valuation représente une priorité. Comme pour les HCSPs, l'objectif consiste à minimiser le nombre de contraintes violées de priorité maximale, puis, à nombre égal, de minimiser le nombre de contraintes violées de priorité inférieure, et ainsi de suite.

Forces et Faiblesses du modèle VCSP

Le formalisme des VCSPs a été intensivement étudié depuis une quinzaine d'années. Les Weighted CSPs ont été utilisés pour modéliser de nombreux problèmes comme l'allocation de ressources [CdGL$^+$99], des problèmes combinatoires d'enchères [San99], ou encore des problèmes de bioinformatique [ZGS08]. Une des grandes forces de ce modèle est liée à la puissance des techniques de filtrage exploitant des mécanismes de transfert de coût le long du réseau de contraintes pour dériver des minorants de très bonne qualité ([CS04, CdGS07, CdGS$^+$08]). De plus, ces minorants sont calculés de manière très efficace ($O(e \times d^2)$ pour l'arc cohérence virtuelle [CdGS$^+$08]).

Toutefois, le modèle VCSP présente quelques faiblesses [Rég04] :

1. *Les coûts sont des fonctions et non des variables* : dans les VCSPs, l'application φ lie un tuple au coût de son insatisfaction (ce coût peut éventuellement être nul). Il n'est donc pas possible d'éliminer des valeurs des domaines en fonction d'une modification de l'objectif.

2. *La loi de composition interne est limitante* : par exemple, il est difficile d'exprimer de manière raisonnable une règle de répartition de l'insatisfaction.
3. *Les contraintes globales ne peuvent pas être utilisées* : les calculs de minorants ne permettent pas directement de prendre en compte des contraintes portant sur plus de trois variables.

3.2.3 La relaxation disjonctive d'un réseau de contraintes

Thierry Petit *et al.* proposent dans [PRB00, Pet02] de transformer la résolution d'un problème sur-contraint (COP) en un problème de satisfaction afin de bénéficier des travaux menés sur les CSPs. Pour cela, les coûts sont intégrés directement au problème via l'ajout d'un ensemble de variables. La version relaxée de chaque contrainte est formulée sous forme d'une disjonction : soit la contrainte est vérifiée et le coût est nul, soit la contrainte est insatisfaite et le coût est précisé. Les règles sur les violations peuvent être alors modélisées par des contraintes portant sur les variables de coût et les variables initiales. Ces contraintes sont qualifiées de « méta-contraintes ».

Soit $R = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un réseau de contraintes, on notera $\mathcal{C} = \mathcal{C}_h \cup \mathcal{C}_s$ avec $\mathcal{C}_h$ l'ensemble des contraintes d'intégrité du problème et $\mathcal{C}_s$ l'ensemble des contraintes de préférence.

Un réseau de contraintes R est transformé en un nouveau réseau de contraintes R' dans lequel le coût de l'insatisfaction de chaque contrainte est représenté par une variable. Le domaine de cette variable est constitué de valeurs numériques positives possédant un élément minimal (représentant le cas où la contrainte est satisfaite), et pouvant être totalement ordonné.

Définition 48 : ***Variable de coût associée à une contrainte [Pet02]***
Soit c_i une contrainte, la variable de coût z_i associée à c_i est une variable à valeurs numériques positives ou nulle telle que :

- *si c_i est satisfaite alors $z_i = 0$;*
- *si c_i est insatisfaite alors $z_i > 0$ (la valeur de z_i quantifie la violation de c_i).*

La valeur de z_i est fonction de la sémantique de la violation qui est associée à la contrainte.

Définition 49 : ***Sémantique de violation***
μ est une sémantique de violation pour une contrainte quelconque $c(X_1, ..., X_n)$ ssi μ est une fonction de $D_{X_1} \times ... \times D_{X_n}$ vers $\mathbb{R}^+$ telle que $\forall \mathcal{A} \in D_{X_1} \times ... \times D_{X_n}$, $\mu(\mathcal{A}) = 0$ ssi $c(X_1, ..., X_n)$ est satisfaite.

Il existe une bijection entre l'ensemble des variables de coût et les contraintes initiales afin qu'une variable de coût ne puisse être associée qu'à une et une seule contrainte.

Définition 50 : ***Contrainte de coût [Pet02]***
Soit c_i une contrainte et z_i sa variable de coût ; la sémantique de violation de c_i est exprimée par une contrainte $\overline{c_i}$ définie sur $\mathcal{X}_{c_i} \cup \{z_i\}$ ($\overline{c_i}$ est appelée contrainte de coût de c_i).

Exemple 6 :
Soit la contrainte binaire c_i définie par $X_1 = X_2$, avec pour domaines $D_{X_1} = D_{X_2} = \{1,2,3\}$. Si l'on choisit comme sémantique de violation la distance entre les deux variables, alors $z_i = |X_1 - X_2|$. Les coûts de violation des différentes instanciations sont présentés dans le tableau 3.2.

X_1	X_2	z_i
1	1	0
1	2	1
1	3	2

X_1	X_2	z_i
2	1	1
2	2	0
2	3	1

X_1	X_2	z_i
3	1	2
3	2	1
3	3	0

TAB. 3.2 – Coûts de violation pour l'exemple 6.

Remarque 5 : Dans les Weighted CSPs, il est possible de modéliser des sémantiques fines en valuant les différents tuples associés à une contrainte. Mais cette représentation va totalement « diluer » la sémantique de violation et la rendre plus difficilement exploitable.

Les contraintes de préférence peuvent être alors modélisées sous forme de contraintes disjonctives en ajoutant ces contraintes de coût au modèle initial. Si une contrainte est satisfaite alors son coût sera nul sinon il sera tel que défini par la sémantique de violation.

Définition 51 : ***Relaxation disjonctive d'une contrainte [Pet02]***
Soit $R = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un réseau de contraintes, $c_i \in \mathcal{C}$ une contrainte et z_i sa variable de coût. La relaxation disjonctive de c_i est la contrainte c_{disj_i} définie par :

$$c_{disj_i} = [[c_i \wedge [z_i = 0]] \vee [\overline{c_i} \wedge [z_i > 0]]$$

Exemple 7 :
La relaxation disjonctive de la contrainte c_i de l'exemple 6 est la suivante :

$$c_{disj_i} = [X_1 = X_2 \wedge z_i = 0] \vee [z_i = \mid X_1 - X_2 \mid]$$

Définition 52 : ***Relaxation disjonctive d'un réseau de contraintes [Pet02]***
Soit $R = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un réseau de contraintes, la relaxation disjonctive de R est un nouveau réseau de contraintes $R' = (\mathcal{X}', \mathcal{D}', \mathcal{C}')$ dérivé de R tel que :

- $\mathcal{X}' = \mathcal{X} \cup \mathcal{X}_\mathcal{Z}$
- $\mathcal{D}' = \mathcal{D} \cup \mathcal{D}_\mathcal{Z}$
- $\mathcal{C}' = \mathcal{C}_h \cup \mathcal{C}_{disj}$

avec $\mathcal{X}_\mathcal{Z} = \{z_1, ..., z_{|\mathcal{C}_{disj}|}\}$, $\mathcal{D}_\mathcal{Z} = \{D_{z_1}, ..., D_{z_{|\mathcal{C}_{disj}|}}\}$, $\mathcal{C}_{disj}$ l'ensemble des contraintes $\mathcal{C}_s$ exprimées sous forme de contraintes disjonctives et $\mathcal{C}_h$ l'ensemble des contraintes d'intégrité.

Exemple 8 :
Soit le CSP $P = \{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$ tel que :
- $\mathcal{X} = \{X_1, X_2, X_3\}$,
- $\mathcal{D} = \{D_{X_1} = \{2,3\}, D_{X_2} = \{1,2\}, D_{X_3} = \{1,2,3\}\}$,
- $\mathcal{C} = \{c_1 = [X_1 > X_2], c_2 = [X_1 = X_3], c_3 = [X_3 > X_2]\}$

La contrainte c_1 est la seule contrainte d'intégrité de cet exemple. La sémantique de violation de la contrainte $(=)$ correspond à la distance entre les valeurs de ses deux variables. La sémantique de violation de la contrainte $(>)$ correspond au carré de l'écart entre les valeurs de ses deux variables. La relaxation disjonctive de P est le réseau de contraintes $P' = (\mathcal{X}', \mathcal{D}', \mathcal{C}')$ tel que :
- $\mathcal{X}' = \mathcal{X} \cup \{z_1, z_2\}$,
- $\mathcal{D}' = \mathcal{D} \cup \{D_{z_1}, D_{z_2}\}$,
- $\mathcal{C}'$= $\{X_1 > X_2$,
  $[X_1 = X_3 \wedge z_1 = 0] \vee [z_1 =\mid X_1 - X_3 \mid]$,
  $[X_3 > X_2 \wedge z_2 = 0] \vee [z_2 = (X_3 - X_2)^2]\}$

Résoudre un CSP sur-contraint n'est plus un problème de satisfaction comme dans le cadre « classique » des CSPs mais un problème d'optimisation de contrainte dont la fonction objectif dépend des variables de coûts. Pour plus de généralité, le critère à optimiser sera défini par un opérateur $\oplus$ quelconque relatif à l'ensemble des variables de coût.

Définition 53 : ***formulation PPC d'un problème sur-contraint [Pet02]***
Soit $R = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un réseau de contraintes et $R' = (\mathcal{X}', \mathcal{D}', \mathcal{C}')$ sa relaxation disjonctive, et $\oplus$ une loi de composition de valeurs entières ou réelles. La formulation PPC du problème sur-contraint défini par P est le problème d'optimisation $P'_{opt} = \{\mathcal{X}'_{opt}, \mathcal{D}'_{opt}, \mathcal{C}'_{opt}\}$ tel que :
- $\mathcal{X}'_{opt} = \mathcal{X}' \cup \{z_{obj}\}$,
- $\mathcal{D}'_{opt} = \mathcal{D}' \cup \{D_{z_{obj}}\}$,
- $\mathcal{C}'_{opt} = \mathcal{C}' \cup \{z_{obj} = \oplus_{i=1}^{|\mathcal{C}_{disj}|} z_i\}$.

Avantages de cette représentation

Le premier avantage de cette représentation est de pouvoir encapsuler les problèmes exprimables par les CSP valués. En effet, en choisissant judicieusement les domaines des variables de coût ainsi que l'opérateur d'agrégation $\oplus$, il est possible de tous les exprimer. Le tableau 3.3 présente comment représenter les différents cadres des CSP valués.

Le second avantage de cette représentation est que les variables de coût permettent d'exprimer de manière explicite et immédiate le contrôle de la violation : équilibrage de la violation, limitation de la violation d'une contrainte, ou encore ajout de contraintes "conditionnelles". L'ajout de contraintes, dite méta-contraintes, portant sur les variables initiales et les variables de coût permettra d'exprimer ce contrôle de la violation.

Sur l'exemple des `NRP`s, on peut souhaiter pénaliser toute heure supplémentaire travaillée et empêcher une infirmière d'avoir plus de cinq heures supplémentaires dans son planning. Pour

Classe	$D(z_i)$	$\oplus$	élément minimum	ordre
Max CSP	{0,1}	$+$	0	$<$
Weighted CSP	$\overline{\mathbb{N}}$	$+$	0	$<$
CSP possibilistes	[0,1]	max	0	$>$
CSP flous	[0,1]	min	1	$<$
CSP lexicographiques	$[0,1]^*$	$\cup$	$\emptyset$	lexicographique

TAB. 3.3 – Encapsulation des paradigmes VCSP.

cela, il suffit d'ajouter une contrainte limitant la valeur maximale de la variable de coût associée à la contrainte pénalisant les heures supplémentaires effectuées par une infirmière.

Un autre point fort de ces méta-contraintes est qu'elles peuvent engendrer des modifications directement sur les domaines des variables de coût (comme c'est le cas dans l'exemple précédent), et par propagation effectuer des retraits de valeur dans les domaines des variables de la contrainte initiale.

3.3 Choix d'un cadre pour la relaxation de contraintes globales

Le cadre des HCSPs peut être immédiatement éliminé car celui-ci impose une modélisation trop contraignante et ne permet pas une expressivité suffisante pour modéliser simplement un grand nombre de propriétés que nous souhaitions pouvoir représenter grâce aux contraintes relaxées.

Le cadre des VCSPs est un cadre très intéressant car il permet d'exprimer de manière fine la violation grâce à l'utilisation « systématique » de contraintes tables. De plus, les algorithmes développés permettent le calcul de minorant de très bonne qualité de manière efficace (`SAC` [CS04], `OSAC` [CdGS07],`VAC` [CdGS$^+$08]).

Le premier problème concernant l'intégration des contraintes globales dans le cadre des VCSPs est que pour mesurer finement la violation il est nécessaire d'utiliser des contraintes tables. Or la représentation sous forme de contrainte table d'une contrainte globale est potentiellement de taille exponentielle. En effet, pour décrire totalement une contrainte globale, il est nécessaire de décrire les d^n instanciations possibles de celle-ci. Or pour l'exemple d'une contrainte globale `AllDifferent`$(\mathcal{X})$ portant sur dix variables dont les domaines contiennent dix valeurs cela correspond à dix milliards d'instanciations.

Le second problème est que les algorithmes proposés pour les VCSPs (pour des cadres dont la loi interne est la somme ; comme c'est la cas pour les Max CSPs et les Weigthed CSPs) utilisent que des contraintes unaires, binaires ou ternaires pour éviter une explosion en temps et en espace. Ceci impose de décomposer les contraintes globales en contraintes binaires. Nous avons vu précédemment que la contrainte globale `AllDifferent`$(\mathcal{X})$ peut être vue comme une conjonction de contraintes binaires de différence. Mais toutes les contraintes globales ne peuvent pas toujours être aussi facilement décomposées en contraintes binaires. Il est parfois nécessaire d'utiliser des méthodes de décomposition plus complexes telles que la représentation duale [DP89] ou

la représentation avec variables cachées [Dec90]. Cependant, même dans le cas parfait de la contrainte `AllDifferent`($\mathcal{X}$), le fait de décomposer la contrainte globale en une conjonction de contraintes binaires équivalentes aura pour conséquence de réduire le niveau de cohérence maintenu et conduira donc à une perte d'un des atouts majeurs des contraintes globales.

Le modèle proposé par Thierry Petit *et al.* possède à la fois la possibilité d'exprimer des mesures de violation fines et de permettre d'exprimer facilement des règles de contrôle de la violation (notamment grâce à l'utilisation de variables pour représenter la violation). De plus cette représentation sous forme de contraintes disjonctive ne possèdent pas de limitation sur les contraintes utilisables (ni sur le nombre de variables, ni sur la nature de la contrainte). Les très bonnes propriétés de ce modèle nous ont conduit à le retenir pour la relaxation de contraintes globales.

Des travaux récents [LL09] permettent l'intégration de certaines contraintes globales relaxées définies dans le cadre des contraintes disjonctives au sein des Weighted CSPs et permettent l'exploitation des algorithmes de filtrage prévus pour ce cadre. Ces travaux seront présentés plus en détail dans le chapitre 7.

3.4 Relaxation de contraintes globales dans le modèle disjonctif

Pour qu'une contrainte globale relaxée ait les mêmes bonnes propriétés qu'une contrainte globale classique (notamment en terme d'efficacité du filtrage), il est nécessaire que cette relaxation tienne compte à la fois :

- de la sémantique de la contrainte,
- de la sémantique de violation.

Ainsi, pour une même contrainte globale, il existera généralement plusieurs versions relaxées selon la sémantique de violation retenue.

Deux sémantiques de violation génériques ont été proposées dans [PRB01] :

- la sémantique de violation basée variables,
- la sémantique de violation basée décomposition (ou basée graphe primal).

Pour illustrer et comparer ces deux sémantiques de violation, nous utiliserons la contrainte globale `AllDifferent`($\mathcal{X}$) comme exemple.

3.4.1 Sémantique de violation basée variables

Pour la sémantique basée variables, la violation engendrée par une instanciation dépend du nombre de variables qu'il est nécessaire de ré-instancier pour satisfaire cette contrainte. Ainsi, pour une contrainte `AllDifferent`($\mathcal{X}$), la violation engendrée par une instanciation ayant quatre variables instanciées à une même valeur sera plus importante que celle ayant deux variables instanciées à une même valeur.

Définition 54 : ***Sémantique de violation basée variable [PRB01]***
Soit c une contrainte globale, la sémantique de violation basée variable μ_{var} mesure pour une instanciation le nombre minimal de variables de sa portée dont il faudrait changer la valeur pour que c soit satisfaite.

Exemple 9 :
Considérons la contrainte `AllDifferent`($\{X_1,X_2,X_3,X_4\}$) avec les domaines suivants $D_{X_1}=\{1\}$ et $D_{X_2}=D_{X_3}=D_{X_4}=\{1,2\}$. Le tableau suivant indique les coûts de violation (pour la sémantique basée variables) de toutes les instanciations possibles.

$\mathcal{A}$				$\mu_{var}(\mathcal{A})$
X_1	X_2	X_3	X_4	
1	1	1	1	3
1	1	1	2	2
1	1	2	1	2
1	1	2	2	2
1	2	1	1	2
1	2	1	2	2
1	2	2	1	2
1	2	2	2	2

Cette définition de la sémantique de violation basée variables a été raffinée pour les contraintes possédant des bornes exprimables à l'aide de variables [BP04].

Considérons la contrainte globale `NumberOfDistinctValues`($N,\mathcal{X}$) qui impose à l'ensemble des variables $\mathcal{X}$ de prendre un nombre de valeurs distinctes égal à N. Si on considère l'instanciation $\mathcal{A}=\{(N=4),(X_1=1),(X_2=1),(X_3=1),(X_4=1),(X_5=1)\}$, alors $\mu_{var}(\mathcal{A})=1$ (il suffit de changer la valeur de N pour que la contrainte soit satisfaite). Or, dans la plupart des applications, les bornes sont fixées et la violation est inhérente aux valeurs prises par les X_i. Dans ce cas, le coût de violation attendu serait de 3.

Définition 55 : ***Raffinement de la sémantique basée variables [BP04]***
Soit $c(\mathcal{X})$ une contrainte globale et une partition de $\mathcal{X}$ en deux ensembles de variables $\mathcal{X}_f$ (variables fixées) et $\mathcal{X}_c$ (variables modifiables). La sémantique de violation basée variables μ_{var} mesure le nombre minimal de variables de $\mathcal{X}_c$ qu'il faudrait changer pour que c soit satisfaite.

Considérons à nouveau l'instanciation $\mathcal{A}=\{(N=4),(X_1=1),(X_2=1),(X_3=1),(X_4=1),(X_5=1)\}$ pour la contrainte globale `NumberOfDistinctValues`($N,\mathcal{X}$) en déclarant, cette fois-ci, que N est une variable fixée. Alors, on obtient le résultat attendu, c'est-à-dire $\mu_{var}(\mathcal{A})=3$.

Remarque 6 : Dans la suite de ce mémoire, lorsque nous parlerons de la sémantique de violation basée variables, nous nous référerons toujours à sa version améliorée.

3.4.2 Sémantique de violation basée décomposition

Dans l'exemple 9, les deux instanciations $\mathcal{A}_1$=$\{(X_1 = 1),(X_2 = 1),(X_3 = 1),(X_4 = 2)\}$ et $\mathcal{A}_2$=$\{(X_1 = 1),(X_2 = 1),(X_3 = 2),(X_4 = 2)\}$ ont le même coût de violation. Or si on regarde le nombre de contraintes binaires de différence issues de la décomposition d'`AllDifferent`($\mathcal{X}$) , on constate que 3 d'entre elles sont insatisfaites par $\mathcal{A}_1$ alors que seulement 2 le sont par $\mathcal{A}_2$.

Exploiter la décomposition d'une contrainte globale en contraintes élémentaires (graphe primal) permet de définir une mesure de violation plus fine que celle basée variables.

Définition 56 : ***Graphe primal [Pet02]***
Le graphe primal Primal(c) d'une contrainte c($\mathcal{X}$) est le graphe tel que :

- *les sommets correspondent aux variables de la contrainte $\mathcal{X}$,*
- *chaque arête représente une contrainte binaire entre deux variables (E_p),*
- *l'ensemble des solutions du réseau de contraintes défini par ($\mathcal{X}$,$\mathcal{D}$,E_p) est identique à l'ensemble des solutions du réseau de contraintes ($\mathcal{X}$,$\mathcal{D}$,{c}).*

Exemple 10 : ***Graphe primal de la contrainte*** `AllDifferent`***($\mathcal{X}$)***
Soit la contrainte globale `AllDifferent`($\{X_1,X_2,X_3,X_4\}$). Le graphe primal de cette contrainte est le graphe complet où chaque arête représente une contrainte binaire de différence $c_{X_iX_j}$ définie sur X_i et X_j.

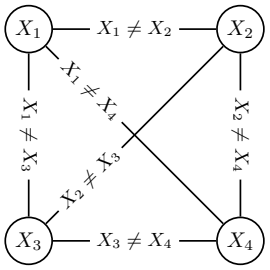


FIG. 3.1 – Graphe primal associé à la contrainte `AllDifferent`($\{X_1,X_2,X_3,X_4\}$)

Définition 57 : ***Sémantique basée décomposition (ou graphe primal) [PRB01]***
Soit c une contrainte globale et Primal(c) son graphe primal. Le sémantique de violation basée décomposition μ_{dec} mesure pour une instanciation le nombre de contraintes binaires insatisfaites dans Primal(c).

Exemple 11 :
Considérons à nouveau la contrainte `AllDifferent`($\{X_1,X_2,X_3,X_4\}$) de l'exemple 9. Le tableau suivant indique les coûts de violation (pour la sémantique basée décomposition) de toutes les instanciations possibles.

$\mathcal{A}$				$\mu_{\mu_{dec}}(\mathcal{A})$
X_1	X_2	X_3	X_4	
1	1	1	1	6
1	1	1	2	3
1	1	2	1	3
1	1	2	2	2
1	2	1	1	3
1	2	1	2	2
1	2	2	1	2
1	2	2	2	3

La sémantique de violation μ_{dec} est moins générale que la sémantique μ_{var} car toutes les contraintes globales ne sont pas décomposables sous la forme d'un graphe primal, comme par exemple la contrainte `Regular`$(\mathcal{X}, \Pi)$.

3.4.3 Autres sémantiques de violation

Différentes sémantiques de violation ont été définies pour des contraintes globales spécifiques. Citons par exemple :

- les sémantiques de violation attachées à la contrainte `Regular`$(\mathcal{X}, \Pi)$ qui sont liées à la distance entre une instanciation des variables de la contrainte et le plus proche mot reconnu par l'automate associé [HPR06].
- la sémantique de violation attachée à la contrainte `cumulative`$(\mathcal{X}, ct, cr, R)$ qui mesure la somme des écarts au carré entre la consommation d'une ressource et la consommation idéale de cette même ressource [PP08].

3.4.4 Consistance et filtrage

Quelque soit c une contrainte et μ une sémantique de violation, il est possible de définir le test de cohérence et l'algorithme de filtrage maintenant la cohérence globale de la façon suivante.

Définition 58 : ***Test de cohérence***
Soit $c(\mathcal{X})$ une contrainte globale, μ une sémantique de violation et z une variable de coût. Soit `soft-`$c(\mathcal{X}, \mu, z)$ *la version relaxée de $c(\mathcal{X})$ selon μ. La contrainte* `soft-`$c(\mathcal{X}, \mu, z)$ *admet une solution ssi il existe une instanciation complète $\mathcal{A}$ des variables de $\mathcal{X}$ telle que $\mu(\mathcal{A}) \leq max(D_z)$.*

Pour une contrainte globale et une sémantique de violation donnée, si l'on peut déterminer l'existence d'une solution en un temps $TEST$ alors il existe un algorithme permettant de maintenir la cohérence globale en faisant appel à ce test un nombre polynomial de fois [BHHW07].

L'algorithme 9 présente une version générique d'un tel algorithme de filtrage permettant de maintenir la cohérence globale en exploitant le test de l'existence d'une solution. Le principe de cet algorithme est de générer pour chaque valeur (X_i, v_j) un sous-problème dans lequel le

Algorithme 9 : Algorithme générique de maintien de la cohérence globale.

```
1 procédure filtrageGAC(c : contrainte, μ : sémantique de violation, z : variable objectif)
2 début
3     pour chaque X_i ∈ 𝒳 faire
4         pour chaque v_j ∈ D_{X_i} faire
5             si ∄𝒜 ∈ D_{X_1}×...×D_{X_{i-1}}×{v_j}×D_{X_{i+1}}...×D_{X_n}, t.q. μ(𝒜) ≤ max(D_z) alors
6                 D_{X_i} ← D_{X_i}\{v_j}
7 fin
```

domaine de la variable X_i est réduit à la valeur v_j et de tester si une solution existe pour ce sous-problème. Si ce n'est pas le cas, alors il est alors possible de filtrer cette valeur.

Donc s'il est possible de tester l'existence d'une solution pour une contrainte globale relaxée en temps $TEST$ alors cette contrainte peut être rendue globalement cohérente en $O(m \times TEST)$. Ainsi s'il est possible de connaître l'existence d'une solution en temps polynomial alors il est possible de maintenir la cohérence globale en temps polynomial.

Il n'est malheureusement pas toujours possible d'effectuer le test d'existence d'une solution en temps polynomial. Mais, nous verrons que dans un tel cas il est possible d'utiliser un algorithme similaire pour maintenir un niveau de cohérence plus faible.

3.5 Conclusion

Dans ce chapitre, nous avons présenté différents cadres permettant la relaxation de contraintes. Nous avons retenu le cadre proposé par Thierry Petit *et al.* permettant d'exprimer simplement des mesures de violation fines et des règles de contrôle de la violation. De plus, aucune limitation n'est posée sur les contraintes utilisables dans ce cadre.

Enfin, relaxer une contrainte globale peut s'effectuer en trois points [Sch05] :

1. Définir une sémantique précise de la violation de la contrainte.
2. Proposer un niveau de cohérence que la contrainte globale relaxée maintiendra.
3. Proposer un algorithme pour maintenir le niveau de cohérence souhaité.

Dans le chapitre suivant, nous présenterons les toutes premières relaxations réalisées dans le cadre disjonctif pour les contraintes globales `AllDifferent`, `Gcc` et `Regular` [Hoe04, HPR06], pour différentes sémantiques de violation.

Chapitre 4

Exemples de contraintes globales relaxées

Sommaire

Depuis le début des années 2000, différentes relaxations ont été proposées pour les contraintes globales `AllDifferent` [Rég94], `Gcc` [Rég96], `Regular` [Pes04], `Same` [Bel00], `cumulative` [BC94] et `Grammar` [QW06]. Même si initialement ces travaux n'ont pas tous été présentés *stricto sensu* dans le cadre de la relaxation disjonctive, ils peuvent être aisément et naturellement formulés dans ce cadre.

Pour pouvoir relaxer une contrainte globale c selon une sémantique de violation μ, il faut à la fois prendre en compte la sémantique de c et celle de μ afin, notamment, d'obtenir un filtrage efficace (pour le niveau de cohérence retenu). Willem-Jan van Hoeve *et al.* [Hoe04, Hoe05, HPR06] ont proposé, pour les contraintes `AllDifferent`, `Gcc`, `Regular` et `Same`, de réutiliser la représentation sous forme de réseau de ces contraintes globales et d'ajouter des arcs, dit de violation, permettant de modéliser et quantifier la violation pour une sémantique donnée. Ces arcs de violation permettent de représenter des excès, des manques ou des transferts de coût. Cette approche n'est pas la seule utilisable. Nous montrerons que dans certains cas il est possible d'exploiter les mêmes algorithmes dans le cadre disjonctif [PRB01].

Dans ce chapitre, nous dressons un état de l'art de la relaxation des trois contraintes globales `AllDifferent`, `Gcc` et `Regular` en nous appuyant sur les travaux de Thierry Petit [PRB01, Pet02] et de Willem-Jan van Hoeve *et al.* [Hoe04, HPR06] qui sont pionniers dans ce domaine. Pour chacune de ces contraintes et pour chaque sémantique de violation retenue μ, nous présentons la version relaxée de c selon la sémantique μ, avec toujours la même progression :

- Définition de la relaxation de c selon μ et arcs de violation à ajouter,
- Test de cohérence et Filtrage (avec les complexités temporelles associées).

Pour la relaxation de `AllDifferent` selon la sémantique de violation basée variables μ_{var}, nous présenterons deux approches : la première, consistant en la recherche de couplages dans un graphe bipartite [PRB01], et la seconde, basée sur la recherche de flots dans un réseau avec arcs de violation [Hoe04].

4.1 Relaxations de la contrainte `AllDifferent`($\mathcal{X}$)

4.1.1 Deux sémantiques de violation

Nous présentons les relaxations de la contrainte `AllDifferent`($\mathcal{X}$) selon les deux sémantiques suivantes :

- la sémantique de violation basée variables μ_{var} (voir Sous-section 3.4.1) ;
- la sémantique de violation basée décomposition μ_{dec} (voir Sous-section 3.4.2).

La première sémantique détermine, pour une instanciation complète, le nombre de variables à ré-instancier pour satisfaire la contrainte.

Définition 59 : ***Sémantique de violation basée variables***
La sémantique de violation basée variables μ_{var} pour la contrainte `AllDifferent`$(\mathcal{X})$ *est définie par :*

$$\mu_{var}(\mathcal{X}) = \sum_{v_j \in Doms} max(|\{i \mid X_i \in \mathcal{X}, X_i = v_j\}| - 1, 0)$$

La seconde sémantique détermine, pour une instanciation complète, le nombre de contraintes binaires de différence insatisfaites.

Définition 60 : ***Sémantique de violation basée décomposition***
La sémantique de violation basée décomposition μ_{dec} pour la contrainte `AllDifferent`$(\mathcal{X})$ *est définie par :*

$$\mu_{dec}(\mathcal{X}) = |\{(X_i, X_j) \mid X_i \in \mathcal{X}, X_j \in \mathcal{X} \text{ t.q. } i < j, X_i = X_j\}|$$

Dans un premier temps, nous présenterons deux mises en œuvre de la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$, qui est la version relaxée de `AllDifferent`$(\mathcal{X})$ selon la sémantique basée variables :

- la première mise en œuvre repose sur la représentation graphe bipartite [PRB01] ;
- la seconde mise en œuvre repose sur le modèle réseau avec arcs de violation [Hoe04].

Dans un second temps, nous présenterons la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$, la version relaxée de `AllDifferent`$(\mathcal{X})$ pour la sémantique basée décomposition. Cette contrainte utilise aussi une représentation réseau/flot avec arcs de violation [Hoe04]. Pour illustrer ces différents travaux, nous utiliserons l'exemple suivant.

Exemple 12 :
Soit le réseau de contraintes $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ avec l'ensemble de variables $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$ qui ont pour domaines D_{X_1}=D_{X_2}=D_{X_3}=$\{1, 2\}$, D_{X_4}=$\{1, 2, 3\}$ et D_z=$\{0, 1\}$ et pour seule contrainte `soft-AllDifferent`$(\mathcal{X}, \mu, z)$.

4.1.2 Relaxation basée variables : approche graphe bipartite

i) Représentation sous la forme « couplage dans un graphe bipartite »

La contrainte globale relaxée `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$ est directement représentée grâce au graphe de valeurs de la contrainte `AllDifferent`$(\mathcal{X})$ (cf. Section 2.2).

ii) Test de cohérence

Tout couplage maximal M dans le graphe de valeurs permet de déterminer le nombre maximal de variables qu'il est possible d'instancier sans générer de violation. M étant maximal, toute variable ne figurant pas dans M devra être ré-instanciée.

Corollaire 4 : ***Test de cohérence de*** `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$
La contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$ *admet une solution* ssi *il existe un couplage maximal M tel que :*

$$|\mathcal{X}| - |M| \leq max(D_z)$$

Ce test de cohérence s'effectue, comme celui de la contrainte `AllDifferent`$(\mathcal{X})$ (voir Section 2.2), grâce à l'algorithme de `Hopcroft & Karp` en $O(\sqrt{n} \times m)$ [HK73].

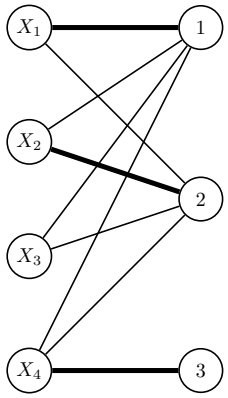


FIG. 4.1 – Couplage maximal de taille 3 associé à l'exemple 12.

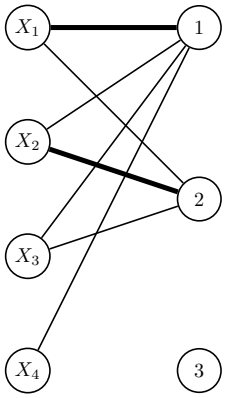


FIG. 4.2 – Graphe de valeurs réduit de $(X_4, 1)$ et couplage maximal associé.

Reprenons l'exemple 12, la figure 4.1 représente un couplage maximal (du graphe de valeurs) de taille 3 (arcs en gras). Comme $|\mathcal{X}|$=4, une seule variable doit être ré-instanciée. Le maximum de la variable de coût z étant de 1, la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$ est cohérente.

iii) Filtrage

Afin de présenter le filtrage, il est nécessaire de définir la notion de graphe de valeurs réduit.

Définition 61 : ***Graphe de valeurs réduit associé à*** (X_i, v_j)
Soit c une contrainte `AllDifferent`$(\mathcal{X})$*, $X_i \in \mathcal{X}$ et $v_j \in D_{X_i}$. Le graphe de valeurs réduit associé à c et à l'instanciation partielle $\{(X_i = v_j)\}$, noté $G_{X_i=v_j}$, est défini par le graphe de valeurs dont toutes les arêtes issues de X_i différentes de $\{X_i, v_j\}$ sont retirées.*

Le test de cohérence peut être utilisé pour réaliser le filtrage comme présenté dans l'algorithme 9 (voir Sous-section 3.4.4). En effet, il suffit de tester s'il existe un couplage maximal $M_{i,j}$ dans le graphe de valeurs réduit $G_{X_i=v_j}$ vérifiant le corollaire 4.

Théorème 4 : ***Viabilité d'une valeur*** (X_i, v_j) ***pour*** `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$
La valeur (X_i, v_j) est viable pour la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$ ssi *il existe au moins un couplage maximal $M_{i,j}$ dans le graphe de valeurs réduit $G_{X_i=v_j}$ t.q.*

$$|\mathcal{X}| - |M_{i,j}| \leq max(D_z)$$

Plutôt que d'effectuer un test de cohérence pour chacune des valeurs, Thierry PETIT *et al.* proposent d'exploiter les résultats présentés dans [Rég95].

Propriété 5 :
Soit un graphe bipartite B et M un couplage maximal de B, notons $M_{i,j}^{max}$ un couplage de cardinalité maximale utilisant l'arc (X_i, v_j). On a alors :

$$|M| - 1 \leq |M_{i,j}^{max}| \leq |M|$$

Ainsi, forcer l'utilisation d'une valeur pour la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$ ne peut provoquer qu'une augmentation de la valuation de 1. Soit M un couplage maximal dans le graphe de valeurs associé à une contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$, alors trois cas sont possibles :

1. $(|\mathcal{X}| - |M|) < max(D_z)$: Toutes les valeurs sont viables.
2. $(|\mathcal{X}| - |M|) = max(D_z)$: Seules les arêtes participant à au moins un couplage maximal représentent des valeurs viables, on peut procéder de la même façon que pour `AllDifferent`$(\mathcal{X})$ pour caractériser ces arêtes (cf. Sous-section 2.2.3) ;
3. $(|\mathcal{X}| - |M|) > max(D_z)$: Aucune valeur n'est viable.

Le filtrage s'effectue en $O(m)$ [Pet02].

Le couplage maximal de la figure 4.1 étant de cardinalité 3, on se retrouve dans la deuxième situation où tout arc ne participant pas à un couplage maximal doit être éliminé. Dans l'exemple 12, les valeurs $(X_4, 1)$ et $(X_4, 2)$ sont non-viables.

4.1.3 Relaxation basée variables : approche réseau avec arcs de violation

i) Représentation sous la forme d'un « réseau avec arcs de violation »

Il est possible d'utiliser la représentation d'`AllDifferent`$(\mathcal{X})$ comme cas particulier d'une contrainte `Gcc`$(\mathcal{X}, l, u)$ en fixant les bornes inférieures à 0 et les supérieures à 1 (voir Sous-section 2.3.4). Willem-Jan VAN HOEVE propose de transformer le réseau associé à la contrainte `AllDifferent`$(\mathcal{X})$, en ajoutant un ensemble d'arcs de violation $\tilde{A}_t$. Ces arcs modélisant le fait qu'une valeur puisse être prise plus d'une fois contre un coût de violation.

À chaque valeur $v_j \in Doms$, on associe un et un seul arc de violation $(v_j \rightarrow t)$ afin de « capturer » et quantifier le coût de violation associé à l'affectation multiple de v_j [Hoe04]. La demande de cet arc de violation est nulle, car idéalement aucune violation ne sera engendrée au niveau de la valeur v_j. La capacité de cet arc est de $\delta^+(v_j) - 1$, car dans le pire toutes les variables pouvant être affectées à v_j le seront et il sera alors nécessaire de ré-instancier $\delta^+(v_j) - 1$ variables.

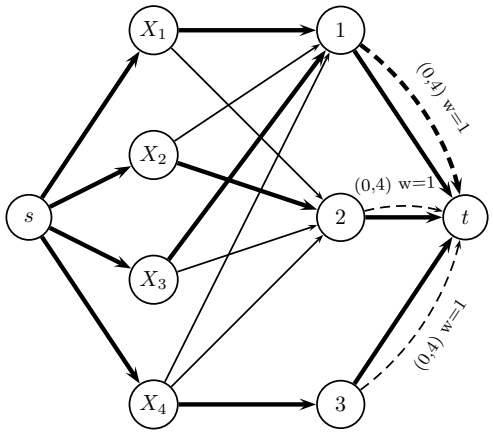


FIG. 4.3 – Flot faisable de valeur n et de poids minimal égal à 1 associé à la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$ de l'exemple 12.

Définition 62 : ***Arcs de violation associés à*** `soft-AllDifferent`***$(\mathcal{X}, \mu_{var}, z)$***

$$\tilde{A}_t = \{(v_j \rightarrow t) \mid v_j \in Doms\}$$
avec
$$\forall a = (v_j \rightarrow t) \in \tilde{A}_t,\ d(a) = 0,\ c(a) = \delta^+(v_j) - 1 \text{ et } w(a) = 1$$

La quantité de flot circulant par un arc de violation $(v_j \rightarrow t)$ est égale au nombre de variables excédentaires ayant pour valeur v_j dans une instanciation $\mathcal{A}$. Comme le poids de cet arc est de 1, le poids du flot passant par celui-ci reflète exactement le coût de violation engendré par ces variables excédentaires.

ii) Test de cohérence

Par construction du réseau, il existe une bijection entre l'ensemble des flots faisables f de valeur n et l'ensemble des instanciations complètes $\mathcal{A}$ des variables de la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{var}, z)$. De plus, le poids d'un flot f faisable de valeur n, $weight(f)$, est égal à la valuation de l'instanciation correspondante $\mathcal{A}$ (c'est-à-dire à $\mu_{var}(\mathcal{A})$).

Corollaire 5 : ***Test de cohérence de*** `soft-AllDifferent`***$(\mathcal{X}, \mu_{var}, z)$***
La contrainte `soft-AllDifferent`*$(\mathcal{X}, \mu_{var}, z)$ admet une solution* ssi *il existe un flot faisable f de valeur n dans le réseau associé de poids $weight(f)$ inférieur ou égal à $max(D_z)$.*

Ainsi, le test de cohérence est effectué par la recherche d'un flot faisable de valeur n de poids minimal. Ce flot est calculable grâce à l'algorithme de `Ford & Fulkerson` [FF56] en $O(n \times m)$. La figure 4.3 présente un flot faisable de valeur 4 et de poids 1 qui correspond à la solution $\{(X_1 = 1),(X_2 = 2),(X_3 = 1),(X_4 = 3)\}$ de coût 1.

iii) Filtrage

Le filtrage proposé dans [Hoe04] est identique à celui proposé par Thierry Petit *et al.* dans [PRB01] et peut donc être effectué en $O(m)$ (voir Sous-section 4.1.2.iii). En effet, il est possible d'utiliser le même constat : la réaffectation d'une variable ne peut provoquer une augmentation du coût de violation que de 1. Ainsi, il est possible de faire apparaître les trois cas vus précédemment.

4.1.4 Relaxation basée décomposition

i) Représentation

Comme pour la sémantique basée variables, Willem-Jan van Hoeve propose d'ajouter au réseau de la contrainte `AllDifferent`$(\mathcal{X})$ des arcs de violation pour modéliser la sémantique de violation μ_{dec}.

À chaque valeur $v_j \in Doms$ est associé $\delta^+(v_j) - 1$ arcs de violation. Ces arcs ont tous une demande nulle et une capacité de 1 (ainsi un arc de violation ne peut être utilisé plus d'une fois). On note chacun de ces arcs de la façon suivante : $(v_j \rightarrow t)_1$,...,$(v_j \rightarrow t)_{\delta^+(v_j)-1}$. Chaque arc a=$(v_j \rightarrow t)_i$ (avec $1 \leq i \leq \delta^+(v_j) - 1$) possède un poids i.

Définition 63 : ***Arcs de violation associés à*** `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$

$$\tilde{A}_t = \{(v_j \rightarrow t)_i \mid v_j \in Doms, i = 1, ..., n\}$$
avec
$$\forall a = (v_j \rightarrow t)_i \in \tilde{A}_t,\ d(a) = 0,\ c(a) = 1 \text{ et } w(a) = i$$

La quantité de violation associée à la valeur v_j correspond à la somme des poids des arcs entre v_j et t utilisés par f. Ainsi, si p variables prennent la valeur v_j dans une instanciation $\mathcal{A}$, alors une unité de flot circulera par les $p-1$ premiers arcs de violation associé à v_j. Le coût de violation associé sera de $\sum_{i=1}^{p} i = \frac{p \cdot (p-1)}{2}$, ce qui correspond au nombre d'arêtes d'une clique à p sommets, et donc au nombre de contraintes binaires insatisfaites associées à la valeur v_j.

La figure 4.4 représente le réseau associé à la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$ de l'exemple 12.

ii) Test de cohérence

Par construction du réseau, il existe une bijection entre l'ensemble des flots faisables f de valeur n et l'ensemble des instanciations complètes $\mathcal{A}$ des variables de la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$. De plus, le poids d'un flot f faisable de valeur n, $weight(f)$, est égal à la valuation de l'instanciation correspondante $\mathcal{A}$ (c'est-à-dire à $\mu_{dec}(\mathcal{A})$).

Corollaire 6 : ***Test de cohérence de*** `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$
La contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$ *admet une solution* ssi *il existe un flot faisable f de valeur n dans le réseau associé de poids $weight(f)$ inférieur ou égal à $max(D_z)$.*

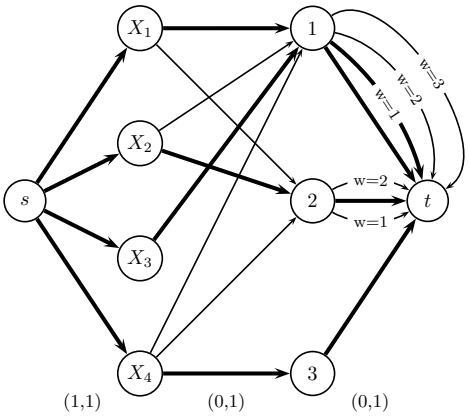


FIG. 4.4 – Flot faisable de valeur n de poids minimal égal à 1 associé à l'exemple 12.

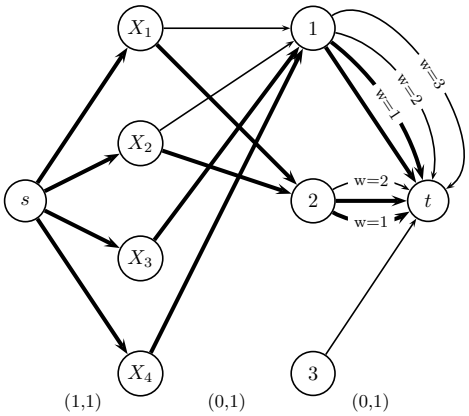


FIG. 4.5 – Flot faisable de valeur n de poids minimal 2 passant par la valeur $(X_4, 1)$ pour l'exemple 12.

Comme pour la sémantique basée variable, le test de cohérence est effectué grâce la recherche d'un flot faisable de valeur n de poids minimal en $O(n \times m)$ (Algorithme de `Ford & Fulkerson` [FF56]). Les arcs en gras dans la figure 4.4 représentent un flot faisable de valeur 4 de poids 1, qui correspond à la solution $\{(X_1=1),(X_2=2),(X_3=1),(X_4=3)\}$ de coût 1.

iii) Filtrage

En construisant l'algorithme de filtrage à partir du test de cohérence, il est nécessaire, pour chaque valeur (X_i, v_j) de trouver un flot faisable f' de valeur n exploitant l'arc $(X_i \rightarrow v_j)$ de poids $weight(f')$ inférieur ou égal à $max(D_z)$.

Comme nous l'avons vu dans le chapitre 1, il est possible d'éviter le calcul complet du flot grâce à la recherche d'un plus court cycle dans le réseau résiduel (voir Propriété 2). Or les arcs provenant du graphe de valeurs $(A_{\mathcal{X}})$ sont de poids nul, donc il suffit de calculer un plus court chemin entre v_j et X_i.

Corollaire 7 : ***Viabilité d'une valeur pour la contrainte*** `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$
Soit la contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$ *et un flot faisable f de valeur n de coût minimal dans le réseau associé. La valeur (X_i, v_j) est viable* ssi *une des deux conditions suivantes est remplie :*

- *l'arc $(X_i \rightarrow v_j)$ appartient au flot f et $weight(f) \leq max(D_z)$;*
- *il existe un chemin p reliant v_j à X_i dans le réseau résiduel du flot f tel que :*

$$weight(f) + weight(p) \leq max(D_z)$$

Une contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$ peut être rendue globalement cohérente en $O(m)$ (voir la preuve dans [Hoe05]). La contrainte `soft-AllDifferent`$(\mathcal{X}, \mu_{dec}, z)$ de l'exemple 12 est globalement cohérente si les valeurs $(X_4, 1)$ et $(X_4, 2)$ sont filtrées.

4.2 Relaxations de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$

4.2.1 Deux sémantiques de violation

Willem-Jan VAN HOEVE *et al.* propose dans [HPR06] de relaxer la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ (voir Section 2.3) selon deux sémantiques de violation :

- la sémantique de violation basée variables (μ_{var}) ;
- la sémantique de violation basée valeurs (μ_{val}) qui comptabilise le nombre de variables créant des manques ou des excès au niveau des bornes[7].

Les deux fonctions suivantes permettent de définir les sémantiques de violation associées à la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$:

- la fonction de manque $s(\mathcal{X}, v_j)$ qui mesure le nombre de variables qu'il faudrait ré-instancier à v_j pour satisfaire la borne inférieure l_j :

$$s(\mathcal{X}, v_j) = max(0, l_j - |\{X_i \mid X_i \in \mathcal{X}, X_i = v_j\}|)$$

- la fonction d'excès $e(\mathcal{X}, v_j)$ qui mesure le nombre de variables instanciées à v_j qu'il faudrait désinstancier pour satisfaire la borne supérieure u_j :

$$e(\mathcal{X}, v_j) = max(0, |\{X_i \mid X_i \in \mathcal{X}, X_i = v_j\}| - u_j)$$

La sémantique de violation basée variables détermine le nombre de variables qu'il est nécessaire de ré-instancier pour satisfaire la contrainte. Ce qui dans le cas de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ revient à éliminer soit le manque sur l'ensemble des bornes inférieures, soit l'excès sur l'ensemble des bornes supérieures. En effet, s'il y a plus de manque que d'excès alors les variables générant des excès peuvent être "absorbées" par les valeurs en manque et inversement (pour plus de détails voir [Hoe05] page 67).

Définition 64 : ***Sémantique basée variables pour la contrainte*** $\texttt{Gcc}(\mathcal{X}, l, u)$ ***[HPR06]*** *Pour une contrainte* $\texttt{Gcc}(\mathcal{X}, l, u)$ *telle que* $\sum_{v_j \in Doms} l_j \leq |\mathcal{X}| \leq \sum_{v_j \in Doms} u_j$*, la sémantique de violation basée variables* μ_{var} *est définie par :*

$$\mu_{var}(\mathcal{X}) = max\left(\sum_{v_j \in Doms} s(\mathcal{X}, v_j), \sum_{v_j \in Doms} e(\mathcal{X}, v_j)\right)$$

La sémantique de violation basée valeurs a été introduite pour mesurer la contribution de chaque violation locale associée à une valeur v_j et non l'aspect global de la violation comme c'est le cas avec la sémantique basée variables. C'est pourquoi, la sémantique basée valeurs mesure la somme des écarts aux bornes l_j et u_j pour toute valeur $v_j \in Doms$.

[7] Nous verrons dans le chapitre 6 que cette sémantique peut être vue comme une sémantique de violation basée décomposition.

Définition 65 : ***Sémantique basée valeurs pour la contrainte*** `Gcc` ***[HPR06]***
Pour une contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$*, la sémantique de violation basée valeurs* μ_{val} *est définie par :*

$$\mu_{val}(\mathcal{X}) = \sum_{v_j \in Doms} (s(\mathcal{X}, v_j) + e(\mathcal{X}, v_j))$$

Dans la suite de cette section, nous verrons comment, à l'aide d'arcs de violation, il est possible d'étendre le réseau de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ à relaxer (voir Sous-section 2.3.2) de telle façon à ce que dans ce réseau étendu un flot faisable f corresponde à une instanciation de coût identique au poids de f.

Pour illustrer les différentes propositions présentées dans cette section, nous utiliserons l'exemple suivant.

Exemple 13 :
Soit le réseau de contraintes $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ avec $\mathcal{X}$=$\{X_1, X_2, X_3, X_4\}$ et D_{X_1}=D_{X_3}= $\{1, 2\}$, D_{X_2}=D_{X_4}=$\{1\}$, $D_z = \{0, 1\}$ et la contrainte $\texttt{soft-Gcc}(\mathcal{X}, [1, 3], [2, 5], \mu, z)$.

Le tableau 4.1 représente toutes les instanciations possibles pour l'exemple 13 ainsi que leurs coûts selon les deux sémantiques de violation.

$\mathcal{X}$	$\sum_{v_j \in Doms} s(\mathcal{X}, v_j)$	$\sum_{v_j \in Doms} e(\mathcal{X}, v_j)$	$\mu_{var}(\mathcal{X})$	$\mu_{val}(\mathcal{X})$
(1,1,1,1)	3	2	3	5
(2,1,1,1)	2	1	2	3
(1,1,2,1)	2	1	2	3
(2,1,2,1)	1	0	1	1

TAB. 4.1 – Table des coûts associés aux instanciations possibles de l'exemple 13.

4.2.2 Sémantique de violation basée variables

i) Représentation

Afin de pouvoir modéliser la violation, un ensemble d'arcs de violation $\tilde{A}_t$ est ajouté au réseau de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ (cf. Section 2.3). Pour tout couple de valeur (v_i, v_j), tel que v_i et v_j appartiennent à $Doms$ et $v_i \neq v_j$, on associe un arc de violation $(v_i \rightarrow v_j)$. Cet arc permet de modéliser la ré-instanciation d'une variable, initialement assignée à v_i, à la valeur v_j [HPR06].

Définition 66 : ***Arcs de violation associés à*** `soft-Gcc` $(\mathcal{X}, l, u, \mu_{var}, z)$

$$\tilde{A}_t = \{(v_i \rightarrow v_j) \mid v_i \in Doms, v_j \in Doms \ t.q. \ i \neq j\}$$
avec
$$\forall a = (v_i \rightarrow v_j) \in \tilde{A}_t, \ d(a) = 0, \ c(a) = \delta^+(v_i) \ et \ w(a) = 1$$

La quantité de flot passant par un arc de violation $(v_i \rightarrow v_j)$ correspond au nombre de variables ayant initialement pour valeur v_i et qui devront être ré-instanciées à la valeur v_j.

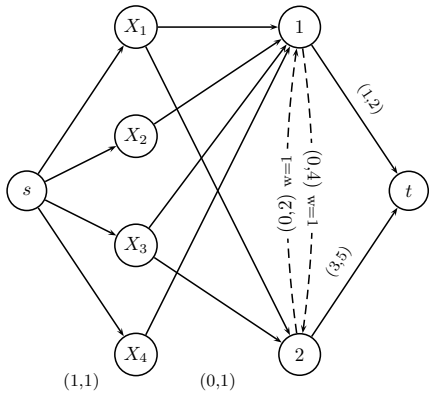


FIG. 4.6 – Réseau associé à la contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{var}, z)$ de l'exemple 13. Les arcs en pointillés représentent les arcs de violation.

Comme le poids de cet arc est de 1, le poids du flot circulant par cet arc correspond exactement au coût de violation engendrée au niveau de la valeur v_i.

La figure 4.6 représente le réseau associé à la contrainte soft-Gcc(X, l, u, μ_{var}, z) de l'exemple 13.

ii) Test de cohérence

Par construction du réseau, il existe une bijection entre l'ensemble des flots faisables f de valeur n et l'ensemble des instanciations complètes $\mathcal{A}$ des variables de la contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{var}, z)$. De plus, le poids d'un flot f faisable de valeur n, $weight(f)$, est égal à la valuation de l'instanciation correspondante $\mathcal{A}$ (c'est-à-dire à $\mu_{var}(\mathcal{A})$).

Théorème 5 : ***Test de cohérence de*** soft-Gcc$(\mathcal{X}, l, u, \mu_{var}, z)$
La contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{var}, z)$ *admet une solution* ssi *il existe un flot faisable f de valeur n dans le réseau associé de poids $weight(f)$ inférieur ou égal à $max(D_z)$.*

Pour réaliser le test de cohérence, il suffit de calculer un flot faisable de valeur n de poids minimal. L'algorithme de Ford & Fulkerson permet de calculer un tel flot grâce au calcul de n plus courts chemins en temps $O(n \times (m+n \cdot log(n)))$ dans le pire cas.

iii) Filtrage

Afin d'éviter d'utiliser le test de cohérence pour tester la viabilité de chaque valeur, il est possible d'utiliser la caractérisation suivante [HPR06].

Corollaire 8 : ***Viabilité d'une valeur (X_i, v_j) pour*** soft-Gcc$(\mathcal{X}, l, u, \mu_{var}, z)$
Soit une contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{var}, z)$ *et f un flot faisable de valeur n et de poids minimal dans le réseau associé, une valeur (X_i, v_j) est viable* ssi *une des deux conditions suivantes est vérifiée :*

– *l'arc $(X_i \rightarrow v_j)$ appartient au flot f et $weight(f) \leq max(D_z)$;*

– *il existe un chemin p reliant v_j à X_i dans le réseau résiduel du flot f tel que*

$$weight(f) + weight(p) \leq max(D_z).$$

Le calcul de la distance d'un sommet à tous les autres peut être réalisé grâce à l'algorithme de `Dijkstra` en $O(m + n \cdot log(n))$. Ainsi, il est possible de rendre globalement cohérente une contrainte `soft-Gcc`$(\mathcal{X}, l, u, \mu_{var}, z)$ en $O(min(n, k) \times (m + n \cdot log(n)))$ [HPR06].

4.2.3 Sémantique de violation basée valeurs

i) Représentation

La sémantique de violation basée valeurs correspond à la somme :
– du cumul des manques de toutes les valeurs $v_j \in Doms$;
– du cumul des excès de toutes les valeurs $v_j \in Doms$.

Pour mesurer l'excès, un ensemble d'arcs de violation $\tilde{A}_{exces}$ est ajouté. Pour chaque valeur $v_j \in Doms$ est associé un et un seul arc de violation $(v_j \rightarrow t)$.

Définition 67 : ***Arcs de violation (excès) associés à*** `soft-Gcc`$(\mathcal{X}, l, u, \mu_{val}, z)$

$$\tilde{A}_{exces} = \{(v_j \rightarrow t) \mid v_j \in Doms\}$$
$$avec$$
$$\forall a = (v_j \rightarrow t) \in \tilde{A}_{exces},\ d(a) = 0,\ c(a) = n \ et\ w(a) = 1$$

La quantité de flot passant par l'arc d'excès $(v_j \rightarrow t)$ sera égale à la différence entre le nombre d'occurrences de v_j dans l'instanciation $\mathcal{A}$ et la borne supérieure u_j, ce qui correspond exactement à $e(\mathcal{A}, v_j)$. Comme le poids de cet arc est de 1 alors le poids du flot circulant par celui-ci correspond exactement à la violation engendrée par l'excès sur la valeur v_j.

Pour mesurer le manque, un ensemble d'arcs de violation $\tilde{A}_{manque}$ est ajouté. À chaque valeur $v_j \in Doms$ est associée un et un seul arc de violation $(s \rightarrow v_j)$.

Définition 68 : ***Arcs de violation (manque) associés à*** `soft-Gcc`$(\mathcal{X}, l, u, \mu_{val}, z)$

$$\tilde{A}_{manque} = \{(s \rightarrow v_j) \mid v_j \in Doms\}$$
$$avec$$
$$\forall a = (s \rightarrow v_j) \in \tilde{A}_{manque},\ d(a) = 0,\ c(a) = l_j \ et\ w(a) = 1$$

La quantité de flot passant par l'arc de manque $(s \rightarrow v_j)$ sera égale à la différence entre la borne inférieure l_j et le nombre d'occurrences de v_j, ce qui correspond à la valeur de $s(\mathcal{A}, v_j)$. De même que pour les arcs d'excès, le poids du flot circulant par cet arc mesure la violation engendrée par le manque sur la valeur v_j.

La figure 4.7 correspond au réseau associé à la contrainte `soft-Gcc`$(\mathcal{X}, l, u, \mu_{val}, z)$ de l'exemple 13.

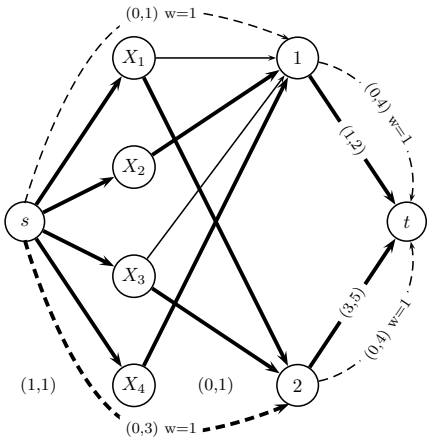


FIG. 4.7 – Réseau associé à la contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{val}, z)$ de l'exemple 13. Les arcs en pointillés représentent les arcs de violation et les arcs en gras un flot faisable.

ii) Test de cohérence

Par construction du réseau, il existe une bijection entre l'ensemble des flots faisables f et l'ensemble des instanciations complètes $\mathcal{A}$ des variables de la contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{val}, z)$. De plus, le poids d'un flot f faisable, $weight(f)$, est égal à la valuation de l'instanciation correspondante $\mathcal{A}$ (c'est-à-dire à $\mu_{val}(\mathcal{A})$).

Corollaire 9 : ***Test de cohérence de*** soft-Gcc$(\mathcal{X}, l, u, \mu_{val}, z)$
La contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{val}, z)$ *admet une solution* ssi *il existe un flot faisable f dans le réseau associé de poids $weight(f)$ inférieur ou égal à $max(D_z)$.*

Remarque 7 : Il est à noter qu'ici le flot faisable n'est pas nécessairement de valeur n. En effet, il est possible que les demandes ne puissent être comblées grâce aux affectations des variables (soit les variables ne sont pas assez nombreuses, soit la structure du réseau rend impossible la satisfaction de toutes les bornes). On peut seulement donner une borne supérieure sur la valeur du flot : $n + \sum_{v_j \in Doms} l_j$. La figure 4.8 présente un cas où le flot est de valeur maximale.

Le test de cohérence peut être réalisé grâce à la recherche d'un flot faisable de poids minimal dans le réseau associé. Ce qui est calculable en $O((n + \sum_{v_j \in Doms} l_j) \times (m + n \cdot log(n)))$ dans le pire cas. Les arcs en gras de la figure 4.7 représentent un flot faisable de valeur 5 et de poids 1 correspondant à la solution $\{(X_1 = 2),(X_2 = 1),(X_3 = 2),(X_4 = 1)\}$ de coût 1.

iii) Filtrage

On peut utiliser la même caractérisation que pour la sémantique de violation basée variables afin d'éviter d'effectuer m fois le test de cohérence.

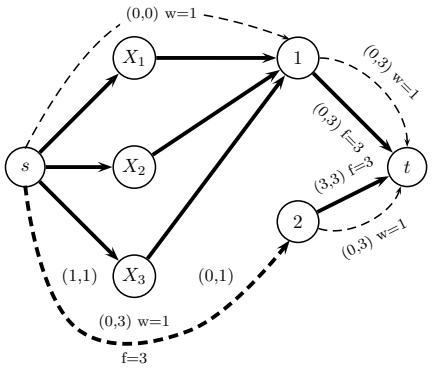


FIG. 4.8 – Flot faisable de poids minimal et de valeur $n + \sum_{v_j \in Doms} l_j$ dans le réseau associé à une contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{val}, z)$.

Corollaire 10 : ***Viabilité de la valeur*** (X_i, v_j) ***pour*** soft-Gcc$(\mathcal{X}, l, u, \mu_{val}, z)$
Soit la contrainte soft-Gcc$(\mathcal{X}, l, u, \mu_{val}, z)$*, un flot* f *faisable de poids minimal dans le réseau associé, une valeur* (X_i, v_j) *est viable* ssi *une des deux conditions suivantes est remplie :*

- *l'arc* $(X_i \rightarrow v_j)$ *appartient au flot* f *et* $weight(f) \leq max(D_z)$ *;*
- *il existe un chemin* p *reliant* v_j *à* X_i *dans le réseau résiduel du flot* f *tel que :*

$$weight(f) + weight(p) \leq max(D_z)$$

Ainsi, la cohérence globale peut être établie en $O(min(n, k) \times (m + n \cdot log(n)))$ [HPR06].

Remarque 8 : En utilisant le même principe que Claude-Guy QUIMPER *et al.*, Alessandro ZANARINI *et al.* améliorent la complexité du test de cohérence ($O(\sqrt{n} \times m)$) et de l'algorithme de filtrage ($O(m + n)$) pour les deux sémantiques μ_{var} et μ_{val}, cela grâce à l'utilisation de deux graphes bipartites plutôt que d'un réseau [ZMP06].

4.3 Relaxations de la contrainte Regular$(\mathcal{X}, \Pi)$

4.3.1 Deux sémantiques de violation

Dans [HPR06], Willem-Jan VAN HOEVE *et al.* proposent, pour la contrainte Regular$(\mathcal{X}, \Pi)$ (voir section 2.4), deux sémantiques de violation associées à deux distances entre mots :

- la distance de Hamming qui permet de définir la sémantique de violation basée variables ;
- la distance d'édition (aussi connue sous le nom de distance de Levenshtein) qui permet de définir la sémantique de violation basée édition.

La distance de Hamming, notée $\mathcal{H}(m_1, m_2)$, mesure pour deux mots de même longueur le nombre de symboles qui différent. La distance d'édition, notée $\mathcal{E}(m_1, m_2)$, mesure le nombre minimal d'opérations élémentaires qu'il est nécessaire d'appliquer pour transformer le mot m_1 en le mot m_2. Ces opérations élémentaires sont l'insertion d'un caractère, l'effacement d'un caractère et la substitution d'un caractère par un autre. Par exemple, la distance de Hamming

entre « chien » et « niche » est de 5 (aucun caractère ne correspond) et la distance d'édition entre ces même mots est égale à 4 (il est nécessaire d'effectuer deux ajouts au début du premier mot et de supprimer deux symboles).

Ces deux sémantiques de violation vont mesurer l'écart, au sens de la distance choisie, entre l'instanciation des variables vue comme un mot et le mot le plus proche d'un langage régulier donné. On a alors les deux définitions suivantes.

Définition 69 : ***Sémantique de violation basée variables pour la contrainte*** `Regular`$(\mathcal{X}, \Pi)$ ***[HPR06]***
Soit une contrainte `Regular`$(\mathcal{X}, \Pi)$*, la sémantique de violation basée variables* μ_{var} *est définie par :*

$$\mu_{var}(\mathcal{X}) = min\{\mathcal{H}(D, \mathcal{X}) \mid D = D_{X_1} \times ... \times D_{X_n} \ t.q. \ D \in L(\Pi)\}$$

Définition 70 : ***Sémantique de violation basée édition pour la contrainte*** `Regular`$(\mathcal{X}, \Pi)$ ***[HPR06]***
Soit une contrainte `Regular`$(\mathcal{X}, \Pi)$*, la sémantique de violation basée édition* μ_{edit} *est définie par :*

$$\mu_{edit}(\mathcal{X}) = min\{\mathcal{E}(D, \mathcal{X}) \mid D = D_{X_1} \times ... \times D_{X_n} \ t.q. \ D \in L(\Pi)\}$$

Pour illustrer le principe du test de cohérence et de l'algorithme de filtrage, nous utiliserons l'exemple suivant.

Exemple 14 :
Soit le réseau de contraintes P=$(\mathcal{X}, \mathcal{D}, \mathcal{C})$ avec $\mathcal{X}$=$\{X_1, X_2, X_3, X_4, X_5\}$ avec pour domaines D_{X_1}=D_{X_2}=D_{X_4}=D_{X_5} = $\{a, b, c\}$, D_{X_3} = $\{a, b\}$, D_z = $\{0, 1\}$ et pour seule contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu, z)$ avec Π l'automate de la figure 4.9.

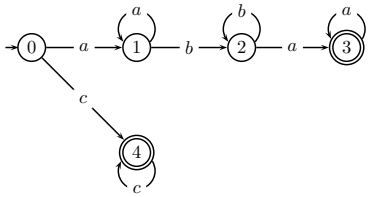


FIG. 4.9 – Automate Π

4.3.2 Sémantique de violation basée variables

i) Représentation

Willem-Jan VAN HOEVE *et al.* proposent d'étendre la représentation sous forme d'un graphe en couches de la contrainte `Regular`$(\mathcal{X}, \Pi)$ (cf. Section 2.4) en ajoutant un ensemble d'arcs de violation $\tilde{A}_{subs}$ pour représenter les substitutions [HPR06].

À chaque transition $\delta(q_l, v_j)$=q_m de l'automate qui devrait être dupliquée dans le graphe en couches, on ajoute un et un seul arc de violation $(q_l^i \rightarrow q_m^{i+1})$ par couche i.

Définition 71 : ***Arcs de violation (substitution) associés à*** `soft-Regular`(X, Π, μ_{var}, z)

$$\tilde{A}_{subs} = \{(q_l^{i-1} \rightarrow q_m^i) \mid \delta(q_l, v_j) = q_m, i = 1, ..., n\}$$
$$avec$$
$$\forall a \in \tilde{A}_{subs},\ w(a) = 1 \text{ et étiqueté par } \Sigma \backslash v_j$$

À chaque fois qu'un arc de violation est utilisé plutôt qu'un arc associé à une transition, alors une variable n'est pas affectée à une valeur "attendue". La sémantique de violation compte le nombre de ces affectations inattendues. Comme le poids des arcs de violation est de 1 (et que tous les autres arcs sont de poids nul), le poids d'un s–t chemin quantifie exactement le nombre de symbole différant du plus proche mot reconnu par l'automate Π.

De la même façon que pour la contrainte `Regular`$(\mathcal{X}, \Pi)$, il est possible de s'affranchir de l'ajout d'arcs "inutiles" grâce à la construction en deux phases d'un graphe en couches réduit (voir Sous-section 2.4.3).

Ce graphe en couches réduit peut être construit comme il suit :

- Durant la phase ascendante, un arc "classique" $(q_l^{i-1} \rightarrow q_m^i)$ étiqueté par v_j est ajouté *ssi* il existe une transition $\delta(q_l, v_j)$=q_m et $v_j \in D_{X_i}$ et si la distance entre s et q_l^{i-1}, notée $dist[q_l^{i-1}]$, est inférieure ou égale à $max(D_z)$. Un arc de substitution $(q_l^{i-1} \rightarrow q_m^i)$ étiqueté par $\Sigma \backslash \{v_j\}$ de poids 1 est ajouté *ssi* il existe une transition $\delta(q_l, v_j)$=q_m et $dist[q_l^i] + 1$ est inférieure ou égale à $max(D_z)$.
- Durant la phase descendante, tous les arcs $(q_l^{i-1} \rightarrow q_m^i)$ non reliés à t sont éliminés en parcourant le graphe dans le sens inverse des arcs.

L'algorithme 10 permet de construire le graphe en couches réduit associé à une contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$.

La figure 4.10 représente le graphe en couches associé à la contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$ de l'exemple 14 après la phase ascendante et la figure 4.11 le même graphe en couches après la phase descendante.

ii) Test de cohérence

Par construction du graphe en couches, il existe une bijection entre l'ensemble des s–t chemins et l'ensemble des instanciations complètes des variables de la contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$. De plus, le poids d'un s–t chemin p, $weight(p)$, est égal à la valuation de l'instanciation correspondante $\mathcal{A}$ (c'est-à-dire à $\mu_{var}(\mathcal{A})$).

Corollaire 11 : ***Cohérence de*** `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$
La contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$ *admet une solution* ssi *il existe un s–t chemin p dans le graphe en couches associé de poids $weight(p)$ inférieur ou égal à $max(D_z)$.*

Le test de cohérence peut être réalisé par le calcul d'un plus court chemin reliant s à t. Comme le graphe en couches est acyclique, le test de cohérence peut être effectué grâce à un parcours en largeur du graphe en couches en $O(n \times |Q| \times |\Sigma|)$.

Algorithme 10 : Construction du graphe en couches réduit pour `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$. [Pes04]

```
1  fonction construireSoftGrapheCouchesReduit(X : variables, Π : automate)
2  début
3      Soit dist un tableau t.q. dist[q_m^i] représente la distance entre s et q_m^i
4      pour m ∈ [0.. |Q|] et i ∈ [0..n] faire dist[q_m^i] ← ∞
5      /* phase ascendante */
6      ajouter arc (s → q_0^0,void)
7      dist[q_0^0] ← 0
8      Soit Fi une file vide
9      Fi ← Fi.enfiler(q_0^0)
10     tant que Fi est non vide faire
11         q_l^i ← Fi.defiler()
12         pour chaque δ(q_l, v_j) = q_m faire
13             ajouter arc (q_l^i → q_m^{i+1},void) de poids 1
14             dist[q_m^{i+1}] ← dist[q_l^i] + 1
15             si v_j ∈ D_{X_{i+1}} alors
16                 (q_l^i → q_m^{i+1}, v_j) de poids 0
17                 dist[q_m^{i+1}] ← dist[q_l^i]
18             si q_m^{i+1} ∉ Fi et dist(q_m^{i+1}) ≤ max(D_z) alors Fi ← Fi.empiler(q_m^{i+1})
19     pour chaque q_l ∈ F et q_l^n possède au moins un arc entrant faire
20         ajouter arc (q_l^n → t,void)
21     /* phase descendante */
22     pour chaque i ∈ {n, .., 1} faire
23         pour chaque m ∈ [0, .., |Q| −1] faire
24             si q_m^i n'as pas d'arc sortant alors
25                 retirer tous les arcs entrant de q_m^i
26     retourner le graphe en couches
27 fin
```

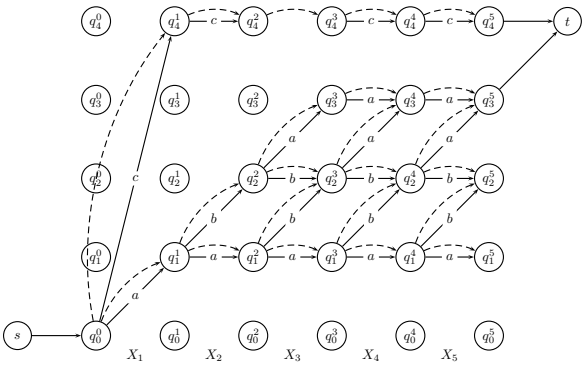


FIG. 4.10 – Graphe en couches à la fin de la phase ascendante de la contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$ de l'exemple 14.

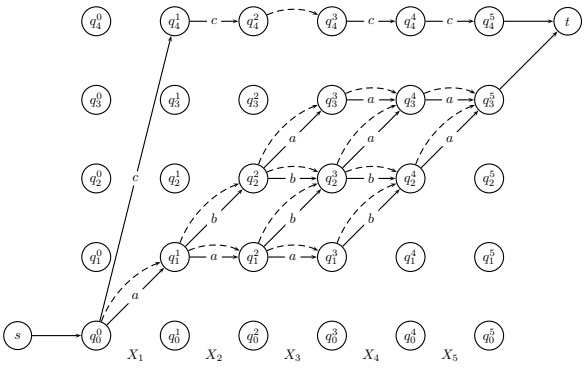


FIG. 4.11 – Graphe en couches à la fin de la phase descendante de la contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$ de l'exemple 14.

De même que pour `Regular`$(\mathcal{X}, \Pi)$, en utilisant un graphe en couches réduit il est possible de réduire le test de cohérence au test de l'existence d'un arc dans A_1.

iii) Filtrage

Le test de cohérence peut être étendu en un algorithme de filtrage, mais impose la calcul de $2 \times m$ plus courts chemins[8]. Dans le graphe en couches réduit, seul les arcs appartenant à au moins un s–t-chemin sont conservés. Pour filtrer les valeurs non viables il suffit, comme pour `Regular`$(\mathcal{X}, \Pi)$, de parcourir chacune des couches du graphe.

Corollaire 12 : ***Viabilité de*** (X_i, v_j) ***pour*** `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$ ***[HPR06]***
Soit une contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$*, la valeur* (X_i, v_j) *est viable* ssi *il existe au moins un arc reliant les couches* $(i-1)$ *et* i *étiqueté par* v_j *dans le graphe en couches associé.*

La cohérence globale peut ainsi être établie par un simple parcours du graphe en couches réduit, c'est-à-dire en $O(n \times |Q| \times |\Sigma|)$ dans le pire cas.

4.3.3 Sémantique de violation basée édition

i) Représentation

Pour la sémantique basée édition, on ne comptabilise pas seulement le nombre de substitutions, mais le nombre de substitutions, d'insertions et d'effacements nécessaire pour transformer un mot m_1 en un mot m_2. C'est pourquoi, aux arcs de substitution $\tilde{A}_{subs}$ présentés pour la sémantique de violation basée variables, on ajoute des arcs de violation pour modéliser les insertions $\tilde{A}_{ins}$ et les effacements $\tilde{A}_{del}$.

[8]Pour un arc $(q_l^{i-1} \rightarrow q_m^i)$, il est nécessaire de calculer un premier plus court chemin reliant s à q_l^{i-1} et un second reliant q_m^i à t.

Définition 72 : ***Arcs de violation (insertion) associés à*** `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$

$$\tilde{A}_{ins} = \{(q_l^i \rightarrow q_m^i) \mid \delta(q_l, v_j) = q_m, i = \{0, ..., n\},\ v_j \in \Sigma\ t.q.\ l \neq m\}$$
$$avec$$
$$\forall a \in \tilde{A}_{ins},\ w(a) = 1$$

Pour simuler l'insertion d'un caractère, un arc reliant deux sommets q_l^i, q_m^i est ajouté. De manière intuitive, les sommets doivent appartenir à la même couche car l'arc ainsi ajouté ne correspond pas au symbole qui sera pris par la variable X_{i+1}, mais à un symbole qu'il faudrait insérer. Les états ainsi reliés doivent correspondre à une transition possible car le symbole inséré est utilisé pour changer l'état de l'automate.

Définition 73 : ***Arcs de violation (effacement) associés à*** `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$

$$\tilde{A}_{del} = \{(q_l^{i-1} \rightarrow q_l^i) \mid i = 1, ..., n\} \backslash A$$
$$avec$$
$$\forall a \in \tilde{A}_{ins},\ w(a) = 1$$

Pour modéliser l'effacement d'un caractère, un arc reliant deux sommets q_l^{i-1}, q_l^i est ajouté. Intuitivement, l'arc représente le fait de ne pas tenir compte du caractère de la i-ième variable.

La figure 4.12 correspond au graphe en couches de la contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$ de l'exemple 14.

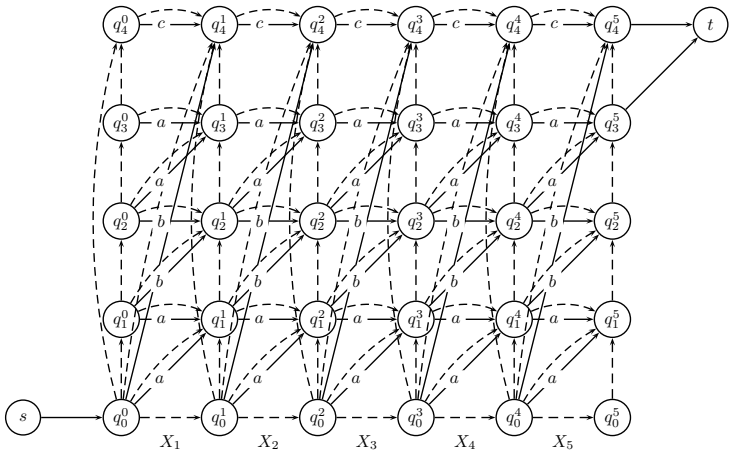


FIG. 4.12 – Graphe en couches de la contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$ de l'exemple 14.

Comme tous les arcs de violation sont de poids non nuls, le poids d'un s–t chemin correspond exactement à la violation engendrée par l'instanciation associé à ce chemin.

ii) Test de cohérence

Par construction du graphe en couches, il existe une bijection entre l'ensemble des s–t chemins et l'ensemble des instanciations complètes des variables de la contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$.

De plus, le poids d'un s–t chemin p, $weight(p)$, est égal à la valuation de l'instanciation correspondante $\mathcal{A}$ (c'est-à-dire à $\mu_{edit}(\mathcal{A})$).

Corollaire 13 : ***Cohérence de*** `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$
La contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$ *admet une solution* ssi *il existe un s–t chemin p dans le graphe en couches associé de poids $weight(p)$ inférieur ou égal à $max(D_z)$.*

Il est possible, qu'au sein d'une couche, des cycles (de poids positifs) se forment. Il n'est plus possible d'utiliser un simple parcours en largeur pour calculer la distance entre s et t.

Pour calculer la distance entre deux sommets, il est nécessaire de casser l'influence de ces cycles. Les cycles sont formés par les arcs associés aux insertions. En parcourant en largeur l'automate Π à partir de chacun des sommets, il est possible de calculer l'influence de ceux-ci en temps $O(|Q| \times |\delta|)$. Les arcs d'insertions de chaque couche sont identiques. Ainsi, il est possible d'utiliser ce prétraitement pour mettre à jour les distances dans le graphe en couches lors de sont déploiement.

Comme pour la sémantique μ_{var}, le déploiement du graphe en couches est réalisé en temps $O(n \times |\delta|)$ (avec $|\delta| = |Q| \times |\Sigma|$). Soit au total un temps de construction du graphe en couches en $O((n+ |Q|) \times |\delta|) = O(n\times |Q| \times |\Sigma|)$ (car il est possible de considérer $|Q| \leq n$ sinon certains états seraient inatteignables [HPR06]).

iii) Filtrage

Pour filtrer les valeurs non-viables, il possible d'utiliser le même principe que pour la sémantique basée variable (μ_{var}).

Corollaire 14 : ***Viabilité d'une valeur pour la contrainte*** `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$
Soit la contrainte `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$*, la valeur (X_i, v_j) est viable* ssi *il existe un s–t chemin p passant par arc reliant les couches $i-1$ et i étiqueté par v_j et de poids $weight(p)$ inférieur ou égal à $max(D_z)$.*

En exploitant le calcul de distances précédent, il est possible d'établir la cohérence globale en $O(n\times |Q| \times |\Sigma|)$ dans le pire cas.

4.4 Conclusion

Dans ce chapitre, nous avons présenté les relaxations selon plusieurs sémantiques des trois contraintes globales : `AllDifferent`, `Gcc` et `Regular`. Pour chacune une modification de la modélisation sous-jacente permet de construire un graphe ou un réseau dans lequel une solution peut être trouvée grâce à l'utilisation d'algorithme de recherche de couplage, de flot ou encore de plus court chemin.

Le tableau 4.2 récapitule pour chaque contrainte présentée dans ce chapitre les différentes sémantiques.

Contrainte	Mesure	Sémantique
`AllDifferent`	Nombre de variables à ré-instancier	μ_{var}
	Nombre de contraintes binaire insatisfaite	μ_{dec}
`Gcc`	Maximum de la somme des manques et de la somme des excès	μ_{var}
	Somme des manques et des excès	μ_{val}
`Regular`	Distance de Hamming au mot le plus du langage reconnu	μ_{var}
	Distance d'édition au mot le plus du langage reconnu	μ_{edit}

TAB. 4.2 – Tableau regroupant pour chaque contrainte présentée ses différentes sémantiques.

On peut constater que les sémantiques présentées ici ne sont pas suffisantes pour modéliser des mesures de violation fines. En effet, avec ces sémantiques il est impossible d'émettre des préférences : que cela soit sur les variables à ré-instancier, sur les bornes d'une contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$, ou sur les différentes transitions d'un automate.

Dans la suite de ce manuscrit, nous présenterons de nouvelles sémantiques de violation permettant l'expression de ces préférences.

Contrainte	Sém.	Représentation	Cohérence	Filtrage	Référence
`soft-AllDifferent`	μ_{var}	Graphe bipartite	$O(\sqrt{n} \times m)$	$O(m)$	[PRB01]
`soft-AllDifferent`	μ_{var}	Réseau	$O(n \times m)$	$O(m)$	[Hoe04]
`soft-AllDifferent`	μ_{dec}	Réseau	$O(n \times m)$	$O(m)$	[Hoe04]
`soft-Gcc`	μ_{var}	Réseau	$O(n \times (m+n \cdot log(n)))$	$O(min(n,k) \times (m+n \cdot log(n)))$	[HPR06]
`soft-Gcc`	μ_{var}	Graphe bipartite	$O(\sqrt{n} \times m)$	$O(m)$	[ZMP06]
`soft-Gcc`	μ_{val}	Réseau	$O((n + \sum_{v_j \in Doms} l_j) \times (m+n \cdot log(n)))$	$O(min(n,k) \times (m+n \cdot log(n)))$	[HPR06]
`soft-Gcc`	μ_{val}	Graphe bipartite	$O(\sqrt{n} \times m)$	$O(m)$	[ZMP06]
`soft-Regular`	μ_{var}	Graphe en couches	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$	[HPR06]
`soft-Regular`	μ_{edit}	Graphe en couches	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$	[HPR06]

TAB. 4.3 – Tableau récapitulatif des contraintes présentées

Deuxième partie

Contributions : De nouvelles contraintes globales relaxées

Chapitre 5

Relaxer `AllDifferent` avec des préférences

Sommaire

Dans ce chapitre, nous proposons deux sémantiques de relaxation pour la contrainte `All-Different` dans le cadre avec préférences. Après avoir défini chacune de ces deux sémantiques, nous caractérisons et mettons en œuvre test de cohérence et filtrage associés.

Pour la première sémantique μ^{Σ}_{var} (basée variables), à chaque variable X_i est associé un poids φ_i qui traduit le coût de sa ré-instanciation. Le coût de violation d'une instanciation complète est la somme des poids des variables à ré-instancier pour satisfaire la contrainte, (et non plus le nombre comme pour la contrainte `soft-AllDifferent` munie de la sémantique μ_{var}). Pour la sémantique μ^{Σ}_{var}, test de cohérence globale et filtrage peuvent être mis en œuvre en temps polynomial grâce aux arcs de violation.

Pour la seconde sémantique μ^{Σ}_{dec} (basée décomposition), à chaque contrainte binaire de différence $X_i \neq X_j$ est associé un poids φ_{ij} qui traduit son coût de violation. Le coût de violation d'une instanciation complète est la somme des poids des contraintes binaires de différence insatisfaites, (et non plus le nombre comme pour la contrainte `soft-AllDifferent` munie de la sémantique μ_{dec}). Pour la sémantique μ^{Σ}_{dec}, nous montrons que le test de cohérence globale est un problème NP-Complet. C'est pourquoi, nous proposons de maintenir une cohérence plus faible en temps polynomial grâce au calcul d'un minorant, dont nous évaluerons expérimentalement la qualité.

5.1 Deux sémantiques de violation avec préférences

5.1.1 Sémantique de violation basée variables μ^{Σ}_{var}

À chaque variable X_i est associé un poids φ_i qui traduit le coût de sa ré-instanciation. Le coût de violation d'une instanciation complète est la somme des poids des variables à ré-instancier pour satisfaire la contrainte.

Si, dans une instanciation $\mathcal{A}$, k variables prennent la même valeur v_j, alors $(k-1)$ de ces variables doivent être ré-instanciées afin de satisfaire la contrainte `AllDifferent`$(\mathcal{X})$. Pour la sémantique μ_{var} ce nombre suffit, mais il n'en est pas de même pour la sémantique μ^{Σ}_{var} qui a besoin de connaître les $(k-1)$ variables à ré-instancier (ou encore, quelle est la variable X_{i_0} qui sera autorisée à prendre la valeur v_j). Afin de minimiser le coût de violation, nous avons choisi, comme variable X_{i_0}, la variable de plus fort poids.

Remarque 9 : X_{i_0} n'est pas forcément unique. En effet, si plusieurs variables sont du même plus fort poids, prendre n'importe laquelle n'influera pas sur la somme des poids des autres variables.

Le coût de violation associé au fait que plusieurs variables peuvent simultanément prendre une même valeur v_j dans une instanciation complète est défini par :

$$w_{reinst}(v_j, \mathcal{X}) = \left(\sum_{X_i \in \mathcal{X} \text{ t.q. } X_i = v_j} \varphi_i \right) - max_{(X_i \in \mathcal{X} \text{ t.q. } X_i = v_j)} \varphi_i$$

Définition 74 : ***Sémantique de violation basée variables avec préférences***
Soit la contrainte `AllDifferent`$(\mathcal{X})$ *et* φ_i *le poids associé à la variable* X_i*, la sémantique de violation basée variables avec préférences* μ^{Σ}_{var} *est définie par :*

$$\mu^{\Sigma}_{var}(\mathcal{X}) = \sum_{v_j \in Doms} w_{reinst}(v_j, \mathcal{X})$$

La sémantique de violation basée variables avec préférences permet de modéliser des problèmes comme celui-ci. Une compagnie aérienne doit attribuer à des passagers (représentés par les variables) des places sur différents vols (ces vols forment le domaine des variables). Malheureusement, le nombre de places disponibles est insuffisant pour permettre à chaque passager de pouvoir partir. Le but est d'attribuer le plus de passagers sur ces vols de telle façon à minimiser les importances de chaque passager sans vol. Ces importances peuvent être fonction de la classe du billet, de la date de réservation du vol, du fait que ce vol ait une correspondance, etc (le coût associé à chaque variable modélisera l'importance du passager associé).

5.1.2 Sémantique de violation basée décomposition μ^{Σ}_{dec}

À chaque contrainte binaire de différence $X_i \neq X_j$ est associé un poids φ_{ij} qui traduit son coût de violation. Le coût de violation d'une instanciation complète est la somme des poids des contraintes binaires de différence insatisfaites.

Définition 75 : ***Sémantique de violation basée décomposition avec préférences***
Soit la contrainte `AllDifferent`$(\mathcal{X})$ *et* φ_{ij} *le poids associé à la contrainte binaire de différence* $X_i \neq X_j$*, la sémantique de violation basée décomposition avec préférences* μ^{Σ}_{dec} *est définie par :*

$$\mu^{\Sigma}_{dec}(\mathcal{X}) = \sum_{1 \leq i < j \leq n \ t.q. \ X_i = X_j} \varphi_{ij}$$

En utilisant la sémantique de violation basée décomposition avec préférences, il est possible de modéliser le problème suivant. Dans ce problème, il est nécessaire d'affecter à chacune des antennes radio d'un parc d'antennes (qui seront représentées par des variables) une fréquence d'émission. Chaque antenne peut émettre sur un ensemble de fréquence (cet ensemble représentera le domaine de la variable associée à l'antenne). Le but est de minimiser les interférences générées par le fait que deux antennes émettent sur la même fréquence. Les interférences peuvent être quantifiées grâce à la distance entre les antennes (et seront modélisées grâce aux coûts associés aux contraintes binaires).

Remarque 10 : Ces deux sémantiques sont des généralisations des sémantiques décrites en section 3.4. En effet, la sémantique de violation μ_{var} peut s'exprimer grâce à la sémantique de violation μ^{Σ}_{var} en fixant à 1 les poids associés aux variables. Il en est de même pour les sémantiques de violation μ_{dec} et μ^{Σ}_{dec} en fixant à 1 les poids associés aux contraintes binaires de différence insatisfaites.

5.2 Relaxation selon la sémantique basée variables

La relaxation de la contrainte globale $\texttt{AllDifferent}(\mathcal{X})$ selon la sémantique de violation basée variables avec préférences μ^{Σ}_{var} est définie comme suit :

Définition 76 : Σ-$\texttt{AllDifferent}(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$
Soit $\mathcal{X}$ un ensemble de variables, W le vecteur des coûts attribués aux variables de $\mathcal{X}$ et z une variable de coût, la contrainte Σ-$\texttt{AllDifferent}(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ admet une solution ssi *il existe une instanciation complète $\mathcal{A}$ tel que :*

$$\mu^{\Sigma}_{var}(\mathcal{A}) \leq max(D_z)$$

Pour illustrer, le test de cohérence et le filtrage d'une contrainte Σ-$\texttt{AllDifferent}(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ nous utiliserons l'exemple suivant.

Exemple 15 :
Soit la contrainte Σ-$\texttt{AllDifferent}(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ avec $\mathcal{X}$=$\{X_1, X_2, X_3, X_4\}$ et D_{X_1}=$\{1\}$,D_{X_2}=D_{X_3}=$\{1, 2\}$, D_{X_4}=$\{2\}$, D_z=[0..8], le vecteur ci-dessous décrit les poids associés aux variables de $\mathcal{X}$.

$\mathcal{X}$	X_1	X_2	X_3	X_4
φ	10	9	1	5

TAB. 5.1 – Vecteur des coûts associés aux variables de l'exemple 15.

5.2.1 Représentation

Le réseau associé à la contrainte Σ-$\texttt{AllDifferent}(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ est identique à celui de la contrainte $\texttt{AllDifferent}(\mathcal{X})$ (représentée sous la forme d'un $\texttt{Gcc}(\mathcal{X}, l, u)$) auquel sont ajoutés des arcs de violation afin de capturer et quantifier la violation définie par μ^{Σ}_{var}.

Définition 77 : ***Arcs de violation associés à*** Σ-$\texttt{AllDifferent}(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$

$$\tilde{A}_t = \{(X_i, t) \mid X_i \in \mathcal{X}\}$$
avec
$$\forall a = (X_i, t) \in \tilde{A}_t,\ d(a) = 0,\ c(a) = 1 \text{ et } w(a) = \varphi_i$$

Si une unité de flot circule dans un de ces arcs (X_i, t) de poids φ_i, cela signifie que la variable associée X_i doit être ré-instanciée contre un coût de violation de φ_i. La somme des poids des arcs de violation traversés par une unité de flot (et donc par construction le poids de ce flot f) est égal à la somme des poids des variables à ré-instancier.

La figure 5.1 représente le réseau associé à la contrainte Σ-$\texttt{AllDifferent}(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ de l'exemple 15.

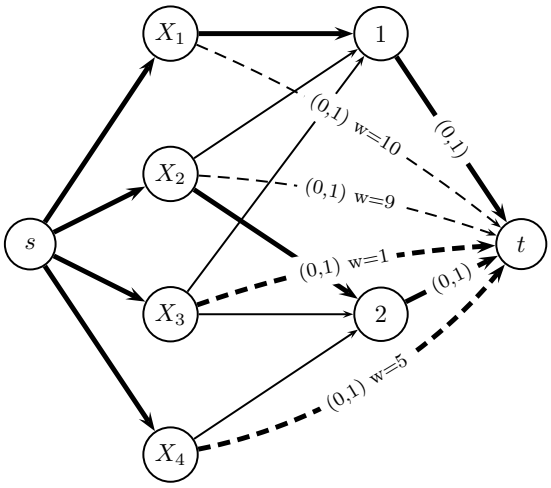


FIG. 5.1 – Réseau associé à l'exemple 15. Les arcs en gras représentent un flot faisable de valeur n de poids minimal.

5.2.2 Test de cohérence

Le test de cohérence pour Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ est réalisé grâce à la recherche d'un flot faisable de valeur n de poids minimal.

Corollaire 15 : ***Cohérence de*** Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$
La contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ *admet une solution* ssi *il existe un flot faisable de valeur* n *de poids minimal* $weight(f)$ *dans le réseau associé tel que* $weight(f)$ *soit inférieur ou égal à* $max(D_z)$.

Preuve 1 :
Par construction, les variables utilisant un arc de violation de la forme $(X_i \rightarrow v_j)$ sont celles autorisées à conserver leur valeur. Les variables utilisant un arc de violation correspondent à des variables devant être ré-intanciées. Toujours par construction, le poids de ce flot correspond exactement à la somme de ces variables à ré-instancier. Donc, si le flot est de poids minimal alors le poids de celui-ci est égal à la somme minimale des poids des variables à ré-instancier, ce qui correspond exactement à la sémantique de violation μ^{Σ}_{var}.

Ainsi, s'il existe un flot faisable f de poids minimal $weight(f)$ dans le réseau associé à la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ tel que $weight(f)$ est inférieur ou égal à $max(D_z)$ alors la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ cohérente. □

Le flot faisable de valeur 4 et de poids 6, décrit par la figure 5.1, représente l'affectation partielle $\{(X_1 = 1),(X_2 = 2)\}$ qui peut être étendue en deux solutions de coût 6 : $\{(X_1 = 1),(X_2 = 2),(X_3 = 1),(X_4 = 2)\}$ et $\{(X_1 = 1),(X_2 = 2),(X_3 = 2),(X_4 = 2)\}$.

Le calcul d'un flot faisable de valeur n de poids minimal peut être réalisé grâce à l'algorithme de `Ford & Fulkerson`, et cela en $O(n \times (m + n \cdot log(n))$.

5.2.3 Filtrage

Il est possible d'établir la cohérence globale, de manière naïve, en appliquant m fois le test de cohérence comme présenté dans l'algorithme 9 (voir Sous-section 3.4.4). Dans le cas de la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$, cela engendre un filtrage relativement coûteux (c'est-à-dire en $O(m \times (n \times (m + n \cdot log(n))))$ dans le pire des cas).

La clef de l'efficacité des algorithmes de filtrage présentés précédemment (voir Chapitres 2 et 4) est d'éviter ce surcoût en ne calculant pour chacune des valeurs qu'un seul plus court chemin dans le réseau résiduel (associé à un flot f lors du test de cohérence).

Dans notre cas, un flot faisable de valeur n de plus petit poids permet d'identifier un ensemble de variables qui garderont leurs valeurs (nous notons cet ensemble $\mathcal{X}_m$). Lors du test de la viabilité de (X_i, v_j), l'instanciation $(X_i = v_j)$ est forcée. Cela ne signifie pas nécessairement que la variable X_i ne devra pas être ré-instanciée (par exemple, s'il y a déjà une variable X_k instanciée à la valeur v_j de poids $\varphi_k > \varphi_i$).

Afin de modéliser, la possibilité de ré-instancier une variable $X_i \in \mathcal{X}_m$, il est nécessaire d'ajouter à l'ensemble des arcs du réseau résiduel (obtenu après le calcul du flot lors du test de cohérence), un nouvel arc de violation.

Durant le test de viabilité de la valeur (X_i, v_j), on ajoute au réseau résiduel un arc connectant v_j à t et de poids égal à φ_i.

Définition 78 : ***Arcs de violation à ajouter lors du test de viabilité de*** (X_i, v_j)

$$\tilde{A}_{X_i} = \{(v_j \rightarrow t)\}$$
$$avec$$
$$\forall a \in A_{X_i},\ d(a) = 0,\ c(a) = 1,\ et\ w(a) = \varphi_i$$

Le plus court chemin reliant v_j à X_i correspond aux modifications à apportées à $\mathcal{X}_m$ lorsque la variable X_i est affecté à v_j.

Corollaire 16 : ***Viabilité d'une valeur*** (X_i, v_j) ***pour*** Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$
Soit la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ *et un flot faisable* f *de valeur* n *et de poids minimal dans le réseau associé. La valeur* (X_i, v_j) *est viable* ssi *une des deux conditions suivantes est satisfaites :*

- *l'arc* $(X_i \rightarrow v_j)$ *appartient à* f *et* $weight(f) \leq max(D_z)$ *;*
- *le plus court chemin* p *reliant* v_j *et* X_i *dans le réseau résiduel de* f *augmenté de l'arc de* $\tilde{A}_{X_i}$ *à un poids* $weight(p)$ *tel que :*

$$weight(f) + weight(p) \leq max(D_z)$$

Le plus court chemin reliant 1 à X_2 est de poids 4 (voir figure 5.2). La valeur $(X_2, 1)$ n'est pas viable et peut être filtrée $(6 + 4 > 8)$. Après le retrait de $(X_2, 1)$ la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ de l'exemple est globalement cohérente.

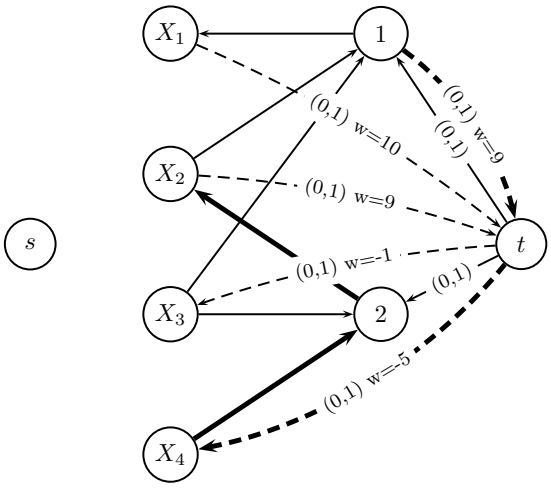


FIG. 5.2 – Réseau associé au test de cohérence de la valeur $(X_2, 1)$. Les arcs en gars représentent un plus court chemin reliant 1 à X_2.

La cohérence globale peut ainsi être établie en calculant pour chaque valeur (X_i, v_j) un plus court chemin reliant v_j et X_i dans le réseau résiduel augmenté. Ceci est réalisable en $O(m \times (m + n \cdot log(n)))$ dans le pire cas.

Remarque 11 : Pour réaliser le filtrage de manière efficace, il est possible d'exclure les variables utilisant un arc de violation dans le flot initial f. En effet, de telles variables peuvent prendre n'importe quelle valeur de leurs domaines sans modifier le coût de l'affectation complète. Par conséquent, si le flot initial f est de poids inférieur ou égal à $max(D_z)$ alors elles sont viables.

5.3 Relaxation selon la sémantique basée décomposition

La relaxation de la contrainte `AllDifferent`$(\mathcal{X})$ selon la sémantique de violation basée décomposition avec préférences μ^{Σ}_{dec} est définie par :

Définition 79 : Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$
Soit $\mathcal{X}$ un ensemble de variables, W une matrice de coûts ($\varphi_{i,j}$ représente le coût de violation associé à la contrainte $X_i \neq X_j$) et z une variable de coût. La contrainte Σ-`All-Different`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ *admet une solution* ssi *il existe une instanciation complète $\mathcal{A}$ t.q. $\mu^{\Sigma}_{dec}(\mathcal{A}) \leq max(D_z)$*

5.3.1 Déterminer l'existence d'une solution est un problème NP-Complet

Théorème 6 :
Déterminer l'existence d'une solution pour une contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ *est un problème NP-Complet.*

Preuve 2 :
Montrons que tout problème de k-coloration des sommets d'un graphe peut être transformé polynomialement en un problème de recherche d'une solution d'une contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$.

Soit un graphe G=(V, E) dont on cherche à k-colorier les sommets. Pour chaque sommet $X_i \in V$, on associe une variable X_i avec D_{X_i}=$\{1..k\}$. On ajoute une contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ portant sur l'intégralité des variables générées comme décrit précédemment. Les poids de la matrice sont définis de la manière suivante : si dans le graphe G, une arête relie les sommets X_i et X_j alors $\varphi_{ij} = 1$ sinon $\varphi_{ij} = 0$. Le domaine de la variable de coût est réduit à la valeur 0.

Une instanciation complète satisfaisant la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ est une k-coloration valide. En effet, chaque variable reçoit une valeur de son domaine et donc chaque sommet une couleur parmi les k. Car $D_z = \{0\}$, aucune contrainte binaire de différence de poids 1 n'est insatisfaite. Par conséquent, toute paire de variables reliées par une telle contrainte prend des valeurs différentes. Ce qui se traduit sur le problème de coloration de graphe par toute paire de sommets reliés par une arête sont de couleurs différentes.

Cette réduction s'effectue en temps polynomial. En effet, il suffit pour chaque paire de sommets de tester s'il existe une arête les reliant. Cette transformation peut donc être faite en $O(n^2)$.
□

La modélisation 2 représente la réduction du problème de 2-coloration de la figure 5.3.

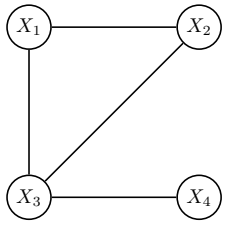


FIG. 5.3 – Graphe à 2-colorier.

Modélisation 2 :

`//ensemble des variables`
Soit $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$ et z.
`//ensemble des domaines`
$D_{X_1} = D_{X_2} = D_{X_3} = D_{X_4} = \{1, 2\}$
$D_z = [0]$
`//ensemble des contraintes`
Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$

W	X_1	X_2	X_3	X_4
X_1		1	1	0
X_2			1	0
X_3				1
X_4				

5.3.2 Une condition suffisante de non-viabilité

Déterminer l'existence d'une solution pour une contrainte Σ-`AllDifferent`$(X, W, \mu^{\Sigma}_{dec}, z)$ étant un problème NP-Complet, nous proposons de maintenir une cohérence locale basée sur le calcul d'un minorant du coût de toute instanciation complète.

Algorithme 11 : Filtrage par minorant

```
1  procédure FiltrageLB(c : contrainte, μ : sémantique de violation, z : variable objectif)
2  début
3      retrait ← true
4      tant que retrait faire
5          retrait ← false
6          pour chaque valeur (X_i, v_j) faire
7              lb_{i,j} ← minorant de {μ(A) | A ∈ D_{X_1}×...×D_{X_{i-1}}×{v_j}×D_{X_{i+1}}...×D_{X_n}}
8              si lb_{i,j} > max(D_z) alors
9                  D_i ← D_i\{v_j}
10                 retrait ← true
11 fin
```

Théorème 7 :
Soit $\mathcal{A}^\star$ une instanciation complète de coût minimal, et lb un minorant de $\mu(\mathcal{A}^\star)$. Si $lb > max(D_z)$ alors il n'existe aucune solution.

Preuve 3 :
Soit $\mathcal{A}$ une instanciation complète quelconque, alors $\mu(\mathcal{A}) \geq \mu(\mathcal{A}^\star)$. De plus, $\mu(\mathcal{A}^\star) \geq lb$ (lb est un minorant de $\mu(\mathcal{A}^\star)$). Si $lb > max(D_z)$, alors $\mu(\mathcal{A}) > max(D_z)$. Donc, si $lb > max(D_z)$ alors $\forall \mathcal{A}, \mu(\mathcal{A}) > max(D_z)$, c'est-à-dire il n'existe aucune solution. □

Nous pouvons alors énoncer une condition suffisante de non-viabilité d'une valeur (X_i, v_j) pour une contrainte Σ-`AllDifferent`$(X, W, \mu^{\Sigma}_{dec}, z)$.

Propriété 6 : *Non-viabilité d'une valeur* (X_i, v_j)
Soit $\mathcal{A}^\star_{i,j}$ telle que $\mu(\mathcal{A}^\star_{i,j}) = min\{\mu(\mathcal{A}) \mid \mathcal{A} \in D_{X_1}\times...\times D_{X_{i-1}}\times\{v_j\}\times D_{X_{i+1}}...\times D_{X_n}\}$. Soit $lb_{i,j}$ un minorant de $\mu(\mathcal{A}^\star_{i,j})$, alors si $lb_{i,j} > max(D_z)$ alors la valeur (X_i, v_j) est non-viable.

Grâce à un test de cohérence basé sur le calcul d'un minorant, il est possible de maintenir une cohérence locale sur le modèle de l'algorithme 9 développé Sous-section 3.4.5. Pour cela, il suffit de tester, pour chaque valeur (X_i, v_j), si le minorant associé $lb_{i,j}$ est strictement supérieur à la valeur maximale de la variable objectif (cf. Algorithme 11). La cohérence maintenue étant locale, il est nécessaire, à la différence de l'algorithme 9, de propager tout retrait. En effet, le retrait d'une valeur (X_i, v_j) peut conduire à une augmentation des minorants d'autres valeurs et donc à d'autres retraits « en cascade ». Enfin, il est clair que la qualité du minorant va influer sur la qualité du filtrage.

Soit MIN le temps nécessaire au calcul du minorant, alors la complexité dans le pire cas de cet algorithme de filtrage est de $O(m^2 \times MIN)$. À chaque passe de calcul m minorants sont calculés et dans le pire cas les valeurs sont retirées une à une.

5.3.3 Un premier minorant basé sur les *conflict-sets*

Un premier minorant lb de la valuation de toute instanciation complète $\mathcal{A}$ peut être calculé grâce à la détection de *conflict-sets* (ensembles de conflits) [PRB01, Jun04].

i) Décomposition d'un ensemble de contraintes en *conflict-sets*

Définition 80 : *Conflict-set*
Soit $P = \{\mathcal{X}, \mathcal{D}, \mathcal{C}\}$ un CSP, K un sous-ensemble de C, $\mathcal{X}_K$ l'ensemble des variables figurant dans les contraintes de K et $\mathcal{D}_K$ l'ensemble de leurs domaines. K est un conflict-set ssi *P_K=$\{X_K, \mathcal{D}_K, K\}$ est incohérent.*

Tout réseau de contraintes peut être décomposé en une union disjointe de *conflict-sets*[9] (éventuellement vide). Dans chaque *conflict-set*, au moins une contrainte est insatisfaite. La somme des poids des contraintes de plus petit poids de chacun de ces *conflict-sets* disjoints constitue un minorant du coût de toute instanciation complète des variables de ce réseau de contraintes [PRB01].

Théorème 8 : ***Minorant basé sur les** conflict-sets*
Soit C un ensemble de contraintes, $\psi = \{\psi_1, ..., \psi_n\}$ une décomposition de C en conflict-sets disjoints et $c^{min}_{\psi_i}$ la contrainte de plus petit poids de ψ_i,

$$lb = \sum_{\psi_i \in \psi} \varphi_{c^{min}_{\psi_i}}$$

La décomposition d'un ensemble de e contraintes en un ensemble de *conflict-sets* disjoints peut être effectuée dans le pire cas en $O(e \times CC)$, avec CC le temps nécessaire à la réalisation du test de cohérence.

Seule la contrainte de plus petit poids est comptabilisée pour chaque *conflict-set*. Il est important de maximiser le poids de cette contrainte de plus petit poids qui servira au calcul du minorant. Heuristiquement, nous testerons les contraintes selon l'ordre décroissant de leur poids[10].

ii) *Conflict-sets* minimaux

Le minorant dépend du nombre de *conflict-sets* présents dans la décomposition. Après chaque détection d'un *conflict-set* CS_i, on propose de minimiser CS_i afin d'améliorer la qualité du minorant.

Définition 81 : *conflict-set **minimal***
Soit K un conflict-set, K est minimal ssi *il n'existe pas de contrainte $c \in K$ telle que $K \backslash \{c\}$ soit un conflict-set.*

[9] Cette décomposition n'est pas nécessairement unique.
[10] Cela permet notamment de maximiser le poids de la plus petite contrainte du premier *conflict-set*.

Algorithme 12 : Calcul de l'ensemble des *conflict-sets* minimaux disjoints

```
1  fonction trouver-un-CS(C : ensemble de contraintes) : un conflict-set
2  début
3      CS ← ∅
4      tant que C ≠ ∅ et CS est cohérent faire
5          Soit c_first la première contrainte de C
6          CS ← CS ∪ {c_first}
7          C ← C\{c_first}
8      si CS est incohérent alors retourner CS
9  fin
10 fonction min-CS(CS : conflict-set) : un conflict-set minimal
11 début
12     C_util ← ∅
13     tant que C_util est cohérent faire
14         C ← C_util, i ←|CS|
15         tant que C est cohérent faire
16             c_last ← la i^ieme contrainte de CS
17             i ← i − 1
18             C ← C ∪ {c_last}
19         C_util ← C_util ∪ {c_last}
20         CS ← C\C_util
21     retourner C_util
22 fin
23 fonction trouver-tous-CS(C : ensemble de contraintes) : un ensemble de conflict-sets
24 début
25     ψ ← ∅
26     tant que C est incohérent faire
27         CS ← min-CS(trouver-un-CS(C))
28         C ← C\CS
29         ψ ← ψ ∪ {CS}
30     retourner ψ
31 fin
```

En minimisant un *conflict-set* CS_i, il est possible de remettre, dans l'ensemble des contraintes restant à décomposer, des contraintes « inutiles » à la détection de CS_i. Ces contraintes peuvent permettre la détection de nouveaux *conflict-sets*, et ainsi, améliorer la qualité du minorant.

La minimisation d'un *conflict-set* contenant ϵ contraintes peut être réalisée en $O(\epsilon^2 \times CC)$ (voir Algorithme 12, fonction `min-CS`). La minimisation repose sur le fait que lors de la détection

d'un *conflict-set*, la dernière contrainte ajoutée a provoqué l'incohérence de l'ensemble de contraintes et est donc une contrainte nécessaire pour la détection de ce *conflict-set*. Ainsi, l'algorithme de minimisation construit une liste de contraintes nécessaires C_{util} en recalculant, de manière successive, le *conflict-set* en ajoutant d'abord les contraintes de C_{util} puis les contraintes restantes du *conflict-set* initial. L'algorithme s'arrête lorsque C_{util} est devenu incohérent, ce qui dans le pire cas arrive quand C_{util} devient identique au *conflict-set* initial.

iii) Application à Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$

Pour Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$, nous proposons de construire un premier minorant de la valuation de toute instanciation complète (étendant une affectation partielle dans le cas du test de viabilité d'une valeur) en calculant les *conflict-sets* disjoints minimaux sur l'ensemble des contraintes binaires de différence. Afin d'améliorer la qualité du minorant, la fonction `trouver-tous-CS` prend en entrée la liste des contraintes triées selon l'ordre décroissant de leur poids.

Nous utilisons comme test de cohérence la cohérence d'arc qui peut-être maintenue en $O(e{\times}d)$ car une contrainte de différence est une contrainte anti-fonctionnelle [HDT92]. De plus, le filtrage peut être fait de manière à économiser le nombre de tests de cohérence en n'insérant dans la file de propagation initiale que la contrainte c_i qui vient d'être ajoutée à l'ensemble à tester. En effet, l'ensemble à tester est arc-cohérent avant l'ajout de c_i (sinon il aurait été caractérisé comme *conflict-set*) et donc si des retraits sont possibles lors de l'ajout de la contrainte c_i c'est cette contrainte qui va générer les premiers retraits.

Pour illustrer cette décomposition en *conflict-sets* disjoints minimaux sur la contrainte Σ-`All-Different`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$, nous utiliserons l'exemple suivant :

Exemple 16 :
Soit la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ avec $\mathcal{X} = \{X_1, ..., X_7\}$, D_{X_1}=$\{2\}$, D_{X_2}=$\{2, 3\}$, D_{X_3}=$\{1, 2, 3\}$, D_{X_4}=D_{X_6}=$\{2, 4\}$, D_{X_5}=$\{3\}$, D_{X_7}=$\{3, 4\}$ et D_z=[0..30]. La matrice de coûts W est définie par le tableau 16.

	X_2	X_3	X_4	X_5	X_6	X_7
X_1	9	8	9	1	9	4
X_2		9	4	8	3	8
X_3			2	1	3	7
X_4				4	7	2
X_5					9	5
X_6						2

TAB. 5.2 – Coûts des contraintes binaires de l'exemple 16.

Sur cet exemple, l'ensemble $\{X_1 \neq X_2,\ X_1 \neq X_4,\ X_1 \neq X_6,\ X_2 \neq X_5,\ X_5 \neq X_6\}$ forme un premier *conflict-set*. Celui-ci peut être minimisé en éliminant les contraintes $X_1 \neq X_4$, $X_1 \neq X_6$,

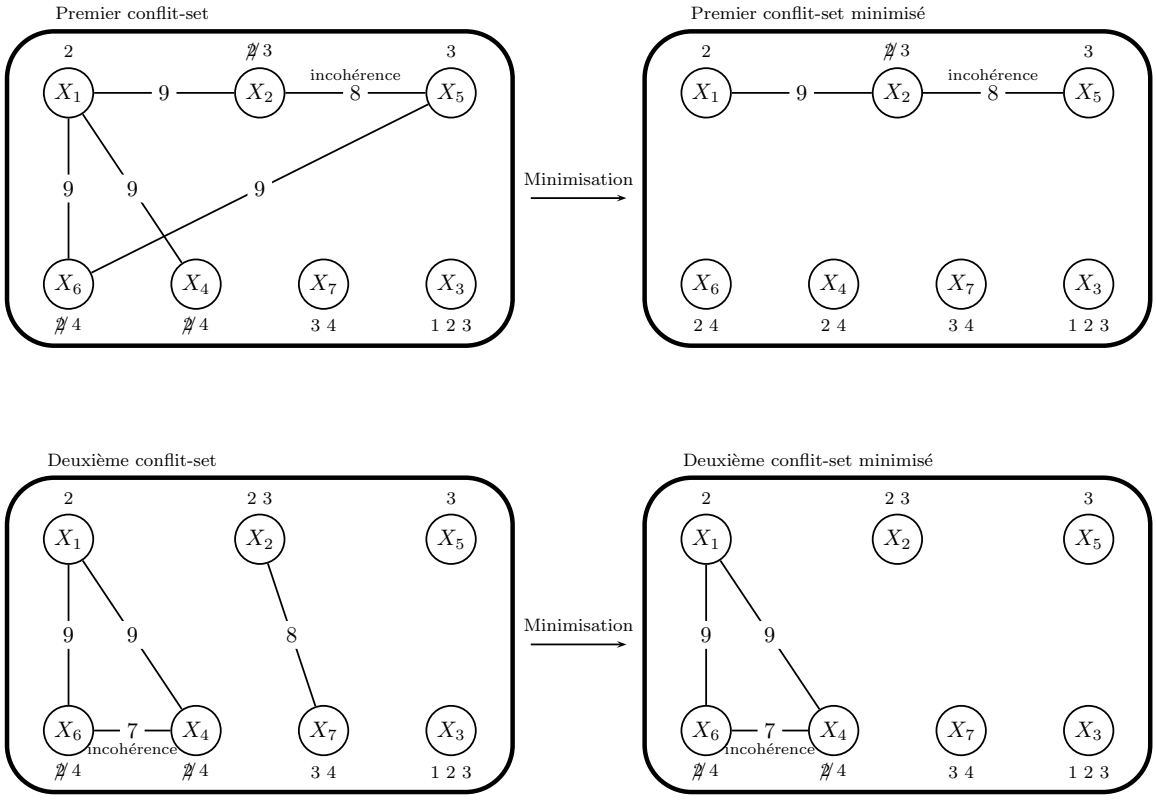


FIG. 5.4 – *Conflict-sets* obtenus pour l'exemple 16.

et $X_5 \neq X_6$. Ainsi on obtient un premier conflict-set minimal $\{X_1 \neq X_2, X_2 \neq X_5\}$. L'ensemble des contraintes $\{X_1 \neq X_4, X_1 \neq X_6, X_2 \neq X_7, X_4 \neq X_6\}$ forme un second *conflict-set*. Celui-ci peut à son tour être minimisé en $\{X_1 \neq X_4, X_1 \neq X_6, X_4 \neq X_6\}$.

Remarque 12 : Le second *conflict-set* n'aurait pas pu être détecté sans l'étape de minimisation du premier *conflict-set*.

La figure 5.4 représente graphiquement cette décomposition en *conflict-sets* minimaux disjoints pour l'exemple 16.

Cette décomposition comporte 2 *conflict-sets* minimaux :

– $\{X_1 \neq X_2, X_2 \neq X_5\}$ de plus petit poids 8,

– $\{X_1 \neq X_4, X_1 \neq X_6, X_4 \neq X_6\}$ de plus petit poids 7.

On obtient ainsi un premier minorant d'une valeur de 15. À titre de repère, le coût optimal d'une solution pour l'exemple 16 est de 18.

5.3.4 Améliorations du minorant

Dans cette section, nous proposons deux améliorations successives du minorant *lb*. La première (*lb*1) repose sur la complétion du minorant à l'aide de contraintes de plus petit poids. La seconde (*lb*2) utilise un raffinement de la complétion en éliminant un sous-ensemble de contraintes qui

seront satisfaites dans une solution de valuation minimale.

i) Compléter à partir du nombre minimal de contraintes insatisfaites

Le nombre de contraintes à relaxer pour une contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ (c'est-à-dire dans le cadre avec préférences) est toujours supérieur ou égal à $\mu^{\star}_{dec}$, au nombre minimal de contraintes binaires à relaxer pour `soft-Alldifferent`$(\mathcal{X}, W, \mu_{dec}, z)$ (c'est-à-dire dans le cadre sans préférences, voir Sous-section 4.1.4).

Théorème 9 :
Pour toute instanciation complète $\mathcal{A}$, soit $C_{unsat}(\mathcal{A})$ l'ensemble des contraintes binaires de différence insatisfaites par $\mathcal{A}$, on a alors :

$$|C_{unsat}(\mathcal{A})| \geq \mu^{\star}_{dec}$$

Preuve 4 :
Soient $\mathcal{A}$ une instanciation complète et $|C_{unsat}(\mathcal{A})|$ le nombre de contraintes binaires insatisfaites par $\mathcal{A}$. Ce nombre est indépendant de la mesure de violation. Soit $\mathcal{A}^{\star}$ une solution optimale pour μ_{dec}, alors $|C_{unsat}(\mathcal{A})| \geq \mu_{dec}(\mathcal{A}^{\star})$ d'où $|C_{unsat}(\mathcal{A})| \geq \mu^{\star}_{dec}$ (par définition de $\mu^{\star}_{dec}$). □

On a bien une inégalité dans le théorème 9, car dans le cadre avec préférences, le critère à minimiser est la somme des poids des contraintes relaxées et non pas leur nombre. L'exemple suivant montre un cas où l'inégalité est stricte.

Exemple 17 :
Soit la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ avec $\mathcal{X}=\{X_1, X_2, X_3, X_4\}$ et pour domaines $D_{X_1}=D_{X_2}=D_{X_3}=D_{X_4} = \{1, 2\}$ et la matrice de coût W suivante :

W	X_1	X_2	X_3	X_4
X_1		1	1	4
X_2			1	4
X_3				4
X_4				

L'instanciation $\mathcal{A}_1=\{(X_1=1),(X_2=1),(X_3=2),(X_4=2)\}$ est optimale au sens de la sémantique μ_{dec} (2 contraintes binaires insatisfaites). Le nombre minimal de contraintes binaires de différence insatisfaites par toute instanciation est donc de 2. L'instanciation $\mathcal{A}_2=\{(X_1=1),(X_2=1),(X_3=1),(X_4=2)\}$ est optimale au sens de la sémantique μ^{Σ}_{dec} or celle-ci insatisfait 3 contraintes binaires.

Soit ψ une décomposition en *conflict-sets* et $\alpha=\mu^{\star}_{dec}-|\psi|$. Si $\alpha > 0$, alors on peut améliorer la qualité du minorant en ajoutant les α contraintes de plus petit poids parmi les restantes.

Définition 82 : *$min\Sigma(E, i)$*
Soit E un ensemble et i un nombre entier positif, l'opérateur $min\Sigma(E, i)$ retourne la somme des i plus petits éléments de E.

Théorème 10 : ***Première amélioration du minorant***
Soit une contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$, C_{dec} l'ensemble des contraintes binaires de différence associées, ψ une décomposition de C_{dec} en conflict-sets minimaux disjoints, C^{min}_{ψ} l'ensemble des contraintes utilisées pour le calcul du minorant lb, $\alpha{=}\mu^{\star}_{dec}{-}|\psi|$. Alors $lb1$ défini comme suit est un minorant de meilleure qualité que lb

$$lb1 = lb + min\Sigma(\{\varphi_i \mid c_i \in C_{dec} \backslash C^{min}_{\psi}\}, \alpha)$$

ii) Ignorer des contraintes satisfaites

Les deux propriétés suivantes décrivent des ensembles de contraintes qui seront satisfaites dans une solution de coût optimal.

La première propriété correspond à une condition nécessaire et suffisante caractérisant le fait qu'une contrainte binaire de différence sera toujours satisfaite C_{sat}.

Propriété 7 :
La contrainte $(X_i \neq X_j)$ est trivialement satisfaite si $D_{X_i} \cap D_{X_j} = \emptyset$

La seconde utilise la structure de la contrainte `AllDifferent`$(\mathcal{X})$ et exploite celle-ci pour caractériser le fait que dans une solution de coût optimal un sous-ensemble de certaines contraintes binaires de différence seront satisfaites.

Propriété 8 : ***Affaiblissement du Théorème de Hall***
Si ω variables impliquées dans un `AllDifferent`$(\{X_1, \ldots, X_n\})$ sont les seules variables à se partager un sous-ensemble de υ valeurs de telle façon à ce que la contrainte `AllDifferent`$(\mathcal{X})$ réduite à ces ω variables et υ valeurs est satisfaite, alors toutes les contraintes binaires de différence portant sur ces ω variables seront satisfaites.

Preuve 5 :
Par hypothèse, il existe un ensemble de valeurs $Val \subset Doms$ tel que :

1. il existe un ensemble de variables $\mathcal{X}_{Val} = \{X_i \in \mathcal{X} \mid \cup_{X_i \in \mathcal{X}} D_{X_i} \supset Val\}$;
2. pour tout $X_i \in (\mathcal{X} \backslash \mathcal{X}_{Val}), D_{X_i} \cap Val = \emptyset$;
3. soit $D'_{X_i} = D_{X_i} \cap Val$, alors la contrainte `AllDifferent`$(\mathcal{X}_{Val})$, dont les domaines des variables de $\mathcal{X}_{Val}$ sont réduits à $D'_{\mathcal{X}_{Val}}$, est satisfaite.

L'ensemble des contraintes binaires de différence peut être décomposé en trois ensembles disjoints :

$$\begin{aligned} C_{dec} = \quad & \{(X_i \neq X_j), X_i \in \mathcal{X}_{val}, X_j \in \mathcal{X}_{val}\} && (E_1) \\ \sqcup \ & \{(X_i \neq X_j), X_i \in \mathcal{X}_{val}, X_j \in \mathcal{X} \backslash \mathcal{X}_{val}\} && (E_2) \\ \sqcup \ & \{(X_i \neq X_j), X_i \in \mathcal{X} \backslash \mathcal{X}_{val}, X_j \in \mathcal{X} \backslash \mathcal{X}_{val}\} && (E_3) \end{aligned}$$

Le calcul de la valuation d'une instanciation peut lui aussi être décomposé :

$$\begin{aligned}\mu(\mathcal{A}) &= \sum_{c_{ij}\in C_{unsat}(C_{dec})} \varphi_{ij} \\ &= \sum_{c_{ij}\in C_{unsat}(E_1)} \varphi_{ij} + \sum_{c_{ij}\in C_{unsat}(E_2)} \varphi_{ij} + \sum_{c_{ij}\in C_{unsat}(E_3)} \varphi_{ij}\end{aligned}$$

Lorsque l'on cherche à minimiser la valuation alors on a :

$$\begin{aligned}min(\mu(\mathcal{A})) &= min(\sum_{c_{ij}\in C_{unsat}(E_1)} \varphi_{ij} + \sum_{c_{ij}\in C_{unsat}(E_2)} \varphi_{ij} + \sum_{c_{ij}\in C_{unsat}(E_3)} \varphi_{ij}) \\ &= min(\sum_{c_{ij}\in C_{unsat}(E_1)} \varphi_{ij}) + min(\sum_{c_{ij}\in C_{unsat}(E_2)} \varphi_{ij}) + min(\sum_{c_{ij}\in C_{unsat}(E_3)} \varphi_{ij})\end{aligned}$$

Car l'union des ensembles E_1, E_2 et E_3 est disjointe et que tous les poids sont positifs. De plus par hypothèse, on sait que :

$$\begin{aligned}min(\sum_{c_{ij}\in C_{unsat}(E_1)} \varphi_{ij}) &= 0 \\ min(\sum_{c_{ij}\in C_{unsat}(E_2)} \varphi_{ij}) &= 0\end{aligned}$$

Donc, on a :

$$min(\mu(\mathcal{A})) = min(\sum_{c_{ij}\in C_{unsat}(E_3)} \varphi_{ij})$$

□

Notons Ω l'ensemble des variables dans le cas décrit par la propriété 8. Alors, l'ensemble des contraintes qui ne seront pas insatisfaites dans au moins une solution de valuation minimale peut être décrit par :

$$C_{Hall} = \{c_{X_iX_j} \mid c_{X_iX_j} \in C_{dec}, X_i \in \Omega\}$$

L'exemple suivant illustre la Propriété 8.

Exemple 18 :
Soit l'ensemble de variables $\mathcal{X} = \{X_1, X_2, X_3, X_4, X_5, X_6\}$ avec D_{X_1}={1,2,}, D_{X_2}={2,3,4}, D_{X_3}={1,3,5} et D_{X_4}=D_{X_5}=D_{X_6}={4,5} et la contrainte `AllDifferent`($\mathcal{X}$). Les variables X_1, X_2 et X_3 peuvent être instanciées respectivement à 1, 2 et 3 sans engendrer de violation. Donc toutes les contraintes où figurent X_1, X_2, ou X_3 seront satisfaites, on a alors $C_{Hall} = \{c_{X_iX_j} \mid i,j \in \{1,2,3\}, i<j\}$ (cf Figure 5.5).

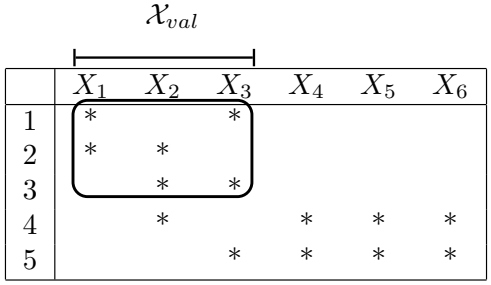


FIG. 5.5 – Représentation graphique de l'exemple 18.

On peut améliorer la qualité du minorant en le complétant uniquement par des contraintes pouvant être insatisfaites. On peut alors exclure les contraintes de C_{sat} et de C_{Hall} de la complétion du minorant.

Théorème 11 : ***Seconde amélioration du minorant***
Soit une contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$*, C_{dec} l'ensemble des contraintes binaires de différence associées, ψ une décomposition de C_{dec} en conflict-sets minimaux disjoints, C^{min}_{ψ} l'ensemble des contraintes utilisées pour le calcul du minorant lb, α=$\mu^{\star}_{dec}$−$|\psi|$. Alors lb2 est défini comme suit :*

$$lb2 = lb + min\Sigma(\{\varphi_i \mid c_i \in (C_{dec} \backslash (C^{min}_{\psi} \cup C_{sat} \cup C_{Hall})\}, \alpha)$$

Pour améliorer le minorant, il est nécessaire de calculer $\mu^{\star}_{dec}$ ce qui est calculable dans le pire des cas en $O(n \times m)$ (voir Section 4.1.4). La propriété 7 peut être testée en $O(e \times d)$ pour toutes les contraintes binaires de C_{dec}, et la propriété 8 en $O(n \times k)$ en se restreigant à un sous-ensemble de ces contraintes (voir annexe A).

La complexité dans le pire des cas de cette phase d'amélioration est de $O(n \times m)$. Ainsi la complexité dans le pire des cas du calcul du minorant est ainsi conservée.

Reprenons l'exemple 16. Le tableau 5.3 indique l'ensemble des contraintes binaires issues de la décomposition de la contrainte `AllDifferent`$(\mathcal{X})$ de cet exemple. Les contraintes en gras représentent les contraintes qui ont été utilisées par le calcul du premier minorant à l'aide des *conflict-sets*. Les contraintes vérifiant la propriété 7 ou la propriété 8 sont hachurées dans ce tableau.

Le nombre minimal de contraintes binaires nécessairement insatisfaites par la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ de l'exemple 16 est de 3. Or, seulement 2 *conflict-sets* ont été trouvés. C'est pourquoi, le poids de la contrainte de plus petit poids non utilisée par le minorant issu de la décomposition en *conflict-set* et n'appartenant pas à C_{sat} ou à C_{Hall} peut être ajouté au minorant.

La contrainte $X_6 \neq X_7$ de poids 2 correspond à cette contrainte. La valeur du nouveau minorant ainsi obtenu est de 17.

5.3.5 Expérimentations

Dans cette section, nous comparons notre approche avec deux méthodes établissant une cohérence locale sur l'ensemble des contraintes binaires issues de la décomposition de Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$:

- `Binary` : une décomposition de la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ en contraintes binaires utilisant la relaxation disjonctive (cf. Chapitre 3). Chaque contrainte binaire $X_i \neq X_j$ est remplacée par une contrainte avec variable de coût $[X_i \neq X_j \wedge z = 0] \vee [X_i = X_j \wedge z = \varphi_{ij}]$.

	X_2	X_3	X_4	X_5	X_6	X_7
X_1	9	~~8~~	9	~~1~~	9	~~4~~
X_2		~~9~~	4	**8**	3	8
X_3			~~2~~	~~1~~	~~3~~	~~7~~
X_4				~~4~~	**7**	2
X_5					~~9~~	5
X_6						2

TAB. 5.3 – Tableau des contraintes binaires de différence disponible pour l'exemple 16 après détection des *conflict-sets* et des contraintes "satisfaites".

– `VAC` : une décomposition de la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ en contraintes binaires dans le cadre des WCSPs et utilisant le minorant fourni par `VAC` [CdGS$^+$08] (il s'agit du calcul de minorant le plus récent proposé pour les WCSPs). Pour nos expérimentations, nous avons utilisé `toulbar2` [dG09].

Pour cela, nous avons effectué des expérimentations sur différentes instances aléatoires pour mesurer la précision du minorant calculé ainsi que la qualité du filtrage induit.

Définition 83 :
La précision du minorant d'une méthode M correspond au rapport entre la valeur du minorant obtenue pour la méthode M et le coût d'une solution minimale.

Définition 84 :
La qualité du filtrage d'une méthode M (maintenant une consistance locale) est mesurée par le rapport entre le nombre de valeurs filtrées grâce à la méthode M et le nombre de valeurs qui auraient dû être filtrées pour établir la cohérence globale.

Afin de calculer la valuation optimale de chaque instance ainsi que la liste des valeurs qu'il est nécessaire de retirer pour établir la cohérence globale, nous avons utilisé une recherche arborescente complète.

Remarque 13 : Les calculs de précision du minorant et de qualité du filtrage ne sont effectués que sur le nœud racine de l'arbre de recherche, c'est-à-dire à l'endroit où le calcul du minorant et le filtrage sont les plus difficiles.

i) Protocole expérimental

Les instances que nous avons générées aléatoirement sont constituées de n variables dont l'union des domaines contient k valeurs. À chaque variable, un domaine est attribué de manière aléatoire parmi l'ensemble de tous les domaines possibles (c'est-à-dire parmi $P_1 \cup P_2 \cup ... \cup P_k$). Une contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ portant sur les n variables de l'instance est ajoutée. Le coût associé à chaque contrainte binaire est déterminé par un tirage aléatoire équiprobable (le poids minimal associé à une contrainte binaire est fixé à 1 et le poids maximal à 10).

Afin d'avoir un ensemble de tests suffisamment large, nous avons fait varier le nombre de variables de n=5 à n=16[11]. Pour chaque valeur du nombre de variables n, nous avons fait varier la taille de l'union des domaines de k=2 (pour obtenir des instances très sur-contraintes) jusqu'à k=n (pour obtenir des instances peu sur-contraintes).

Remarque 14 : Tous les résultats reportés dans la suite de cette sous-section sont des moyennes sur 50 instances.

Le rapport k/n nous semble être une bonne mesure du caractère sur-contraint d'une contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$. En effet, plus ce rapport est petit et plus le nombre de valeurs disponibles est faible par rapport au nombre de variables présentes dans la contrainte.

ii) Précision du minorant

Dans un premier temps, on fixe quatre valeurs pour le ratio k/n et on étudie l'évolution de la précision du minorant, en fonction du nombre n de variables, pour chacune des trois méthodes étudiées (cf. Figures 5.6, 5.7, 5.8 et 5.9).

On constate sur ces quatre premières figures que le minorant produit par `Binary` est de qualité beaucoup plus faible que ceux produits à l'aide de `VAC` ou de la contrainte Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$. Ce résultat n'est pas en soit surprenant. En effet, la relaxation disjonctive n'utilise aucune information sur la structure du problème ni technique de transfert de coûts.

Pour `VAC` et Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$, lorsque l'instance est très sur-contrainte (k/n=0, 2 voir la figure 5.6) alors les minorants obtenus sont très proches. En revanche, lorsque les instances deviennent moins sur-contraintes, alors le minorant calculé par Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ est de meilleure qualité que celui obtenu par `VAC`. En effet, le minorant pour Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ n'utilise pas seulement l'information obtenue via la détection d'incohérence. La complétion tire partie de la structure du problème et permet ainsi d'améliorer de manière significative la qualité du minorant lorsque peu de *conflict-sets* sont détectés.

Dans un second temps, on s'intéresse à l'évolution de la précision du minorant fourni par Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ en fonction du ratio k/n (cf. Figure 5.10). On constate l'existence de trois phases distinctes :

- $k/n < 0.4$: le problème est très sur-contraint. La détection de *conflict-sets* est très efficace et permet ainsi d'obtenir un minorant très informé.
- $0.4 \leq k/n < 0.6$: le problème est moins sur-contraint. La détection de *conflict-sets* ne permet plus d'obtenir un minorant aussi bien informé, et la complétion ne permet pas d'améliorer suffisamment la qualité du minorant.
- $0.6 \leq k/n$: le problème est peu sur-contraint. Il est difficile de détecter des incohérences grâce aux contraintes binaires sous-jacentes. En revanche, la complétion devient suffisamment performante pour permettre au minorant de préserver une bonne précision. Par

[11] La recherche arborescente, utilisée pour déterminer les valuations optimales et les valeurs à filtrer, est de complexité temporelle exponentielle, ce qui explique que nous avons fixé le nombre maximal de variables à 16.

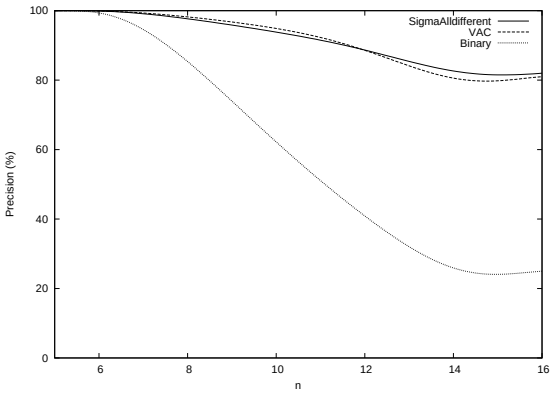


FIG. 5.6 – Précision du minorant pour k/n=0, 2.

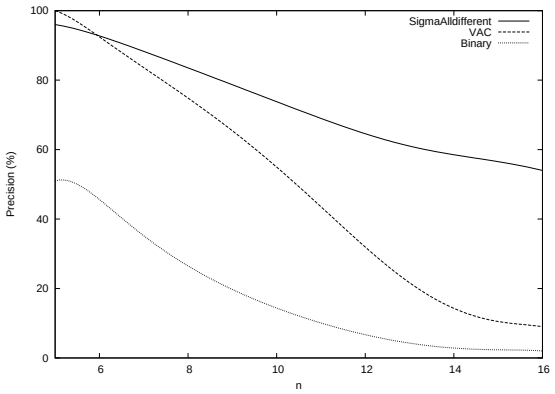


FIG. 5.7 – Précision du minorant pour k/n=0, 4.

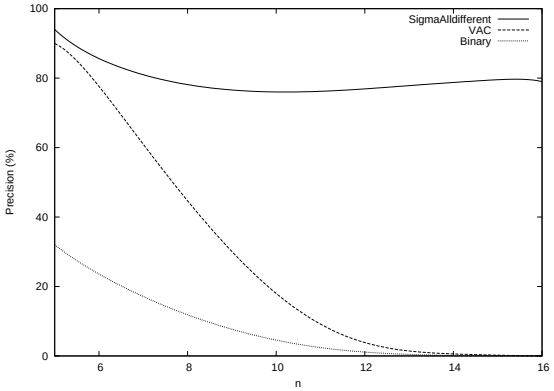


FIG. 5.8 – Précision du minorant pour k/n=0, 6.

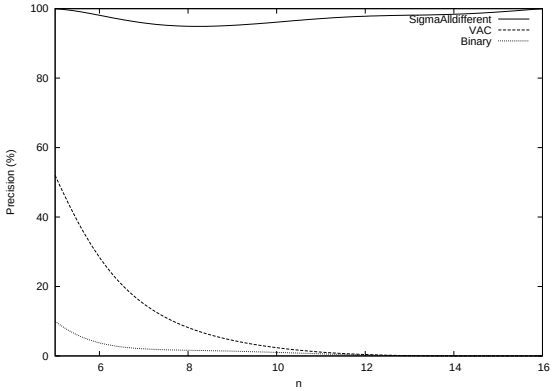


FIG. 5.9 – Précision du minorant pour k/n=0, 8.

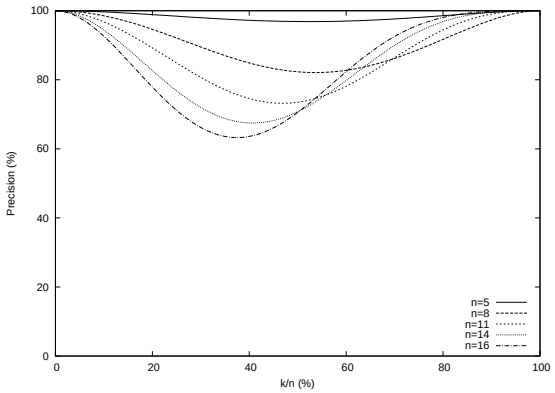


FIG. 5.10 – Précision du minorant calculé par Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ en fonction de k/n.

définition, une solution optimale insatisfera les contraintes de plus petit poids et de par ce fait la complétion permet d'obtenir un bon minorant en reproduisant ce principe.

iii) Qualité du filtrage

Dans un premier temps, on fixe quatre valeurs pour le ratio k/n et on étudie l'évolution de la qualité du filtrage, en fonction du nombre n de variables, pour chacune des trois méthodes étudiées (cf. Figures 5.11, 5.12, 5.13 et 5.14).

Remarque 15 : Le filtrage de `VAC` et celui de Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ sont mis en œuvre de manière similaire. Pour chaque valeur (X_i, v_j), on génère une instance dans laquelle le domaine de X_i est réduit à v_j. Pour cette instance, si le minorant obtenu est supérieur à $max(D_z)$ alors (X_i, v_j) est non-viable et donc retirée.

La qualité du filtrage de `Binary` est assez faible par rapport à la qualité du filtrage de `VAC` ou la qualité du filtrage de Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$. Dans le cas très sur-contraint le filtrage de `VAC` est de meilleure qualité que celui de Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$. Lors du test de viabilité d'une valeur (X_i, v_j) le problème devient un peu plus sur-contraint (car toutes les autres valeurs de D_{X_i} sont retirées). Or comme semble le montrer l'étude la précision, plus le problème est sur-contraint et plus la précision du minorant fourni par `VAC` est bonne. Ceci influence directement la qualité du filtrage fourni par `VAC`. En revanche dès que le problème devient moins sur-contraint le filtrage de Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ devient légèrement meilleur à celui de `VAC`. De manière générale, la qualité du filtrage devient assez faible dès que $n \geq 10$, cela peu importe la méthode utilisée.

La figure 5.15 représente l'évolution de la qualité du filtrage de Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ en fonction du ratio k/n. La qualité du filtrage étant très dépendante de la précision du minorant, le comportement obtenu est identique à celui décrit Figure 5.10.

iv) Étude de l'apport de la complétion

La figure 5.16 présente l'évolution de la quantité de contraintes issues de la complétion ajoutées au minorant par rapport au nombre total de contraintes utilisées pour calculer celui-ci. On constate clairement que lorsque le ratio k/n augmente alors la proportion de contraintes ajoutées au minorant provenant de la complétion devient importante. Pour $n = 16$, la complétion apporte plus de 80% des contraintes utilisées dès que le ratio k/n est supérieur à $0,4$.

Naturellement, on retrouve le même constat pour la proportion du minorant (en qualité) provenant de la complétion (voir figure 5.17).

5.3.6 Discussions

La faiblesse principale dans le calcul du minorant est liée au faible niveau de cohérence qui est utilisé lors de la détection des *conflict-sets* (c'est-à-dire la cohérence d'arc).

Pour améliorer la qualité du minorant, une première possibilité serait d'utiliser durant la phase de détection des *conflict-sets* un niveau de cohérence plus élevé comme par exemple la cohérence d'arc singleton [BD05]. Une seconde possibilité consisterait à essayer de faire ap-

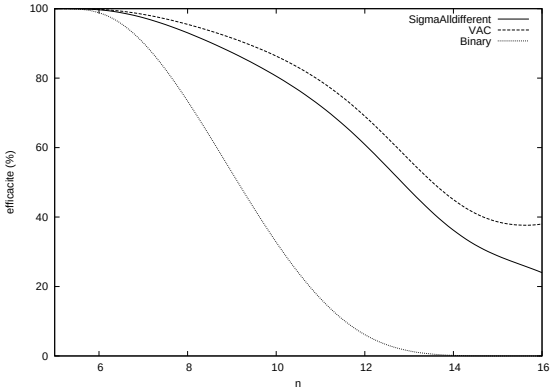


FIG. 5.11 – Qualité du filtrage pour k/n=0, 2.

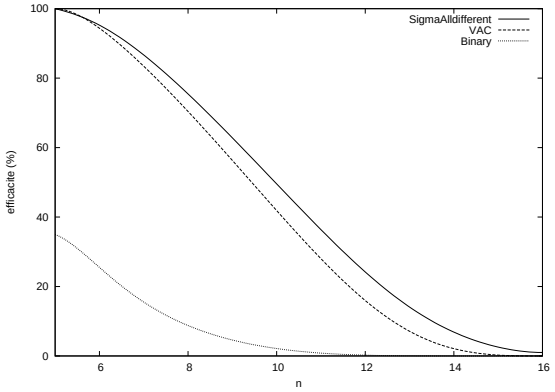


FIG. 5.12 – Qualité du filtrage pour k/n=0, 4.

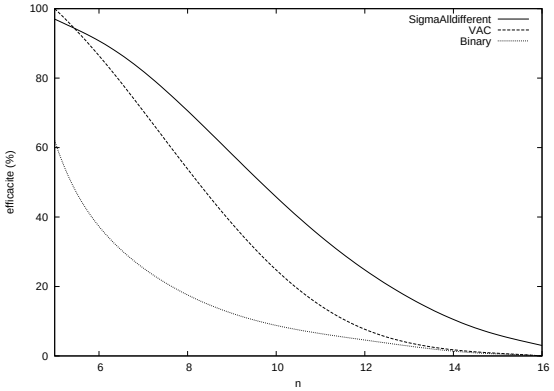


FIG. 5.13 – Qualité du filtrage pour k/n=0, 6.

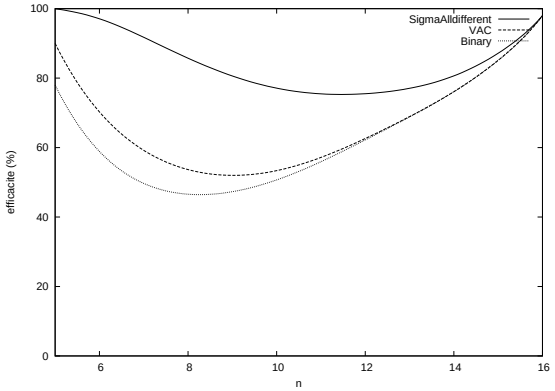


FIG. 5.14 – Qualité du filtrage pour k/n=0, 8.

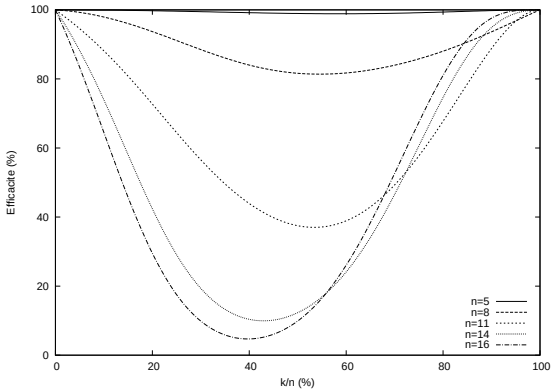


FIG. 5.15 – Qualité du filtrage de Σ-`AllDifferent` en fonction de k/n.

paraître des contraintes `AllDifferent`($\mathcal{X}$) dans l'ensemble des contraintes binaires à tester. En effet, à chaque ajout d'une contrainte binaire dans l'ensemble des contraintes binaires de différence, il suffirait de tester si cette contrainte complète une clique de contrainte binaire, et si cela est le cas remplacer cette clique par une contrainte `AllDifferent`(X) équivalente.

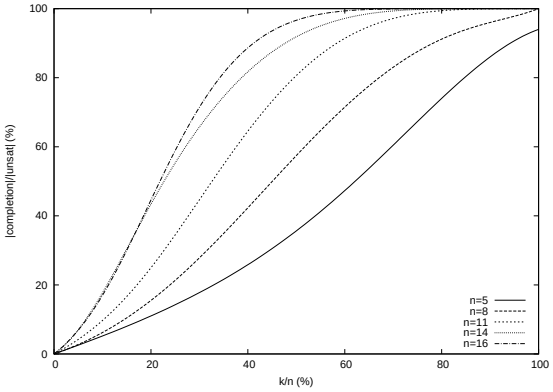


FIG. 5.16 – Proportion du nombre de contraintes ajoutées au minorant grâce à la complétion.

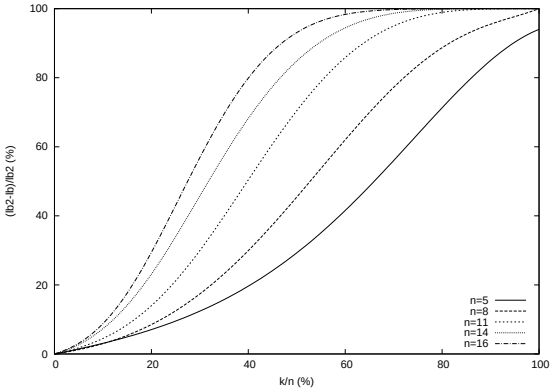


FIG. 5.17 – Proportion du poids des contraintes ajoutées au minorant grâce à la complétion.

5.4 Conclusions

Dans ce chapitre nous avons proposé pour Σ-`AllDifferent`$(\mathcal{X}, W, \mu^{\Sigma}_{dec}, z)$ deux sémantiques de violation avec préférences (voir tableau 5.4).

La première sémantique, μ^{Σ}_{var}, mesure la somme des poids des variables qu'il est nécessaire de ré-instancier pour satisfaire la contrainte `AllDifferent`$(\mathcal{X})$. La seconde sémantique, μ^{Σ}_{dec}, mesure la somme des poids des contraintes binaires insatisfaites dans la décomposition en contraintes binaires de différence de la contrainte `AllDifferent`$(\mathcal{X})$. Ces deux sémantiques sont des généralisations des sémantiques proposées dans [Hoe04].

Contrainte	Mesure	Sémantique
`AllDifferent`	Somme des poids des variables à ré-instancier	μ^{Σ}_{var}
	Somme des poids des contraintes binaires de différence insatisfaites	μ^{Σ}_{dec}

TAB. 5.4 – Tableau récapitulant les sémantiques de violation pour Σ-`AllDifferent`.

Dans le cas de la sémantique basée variables avec préférences, nous avons proposé un algorithme permettant d'établir la cohérence globale en temps polynomial. En revanche dans le cas de la seconde sémantique, nous avons, dans un premier temps, prouvé que le fait de tester la cohérence globale est un problème NP-Complet. C'est pourquoi dans un second temps, nous avons proposé un algorithme permettant de maintenir un niveau de cohérence plus faible que la cohérence globale reposant sur le calcul de *conflict-set*. Le tableau 5.5, résume pour chacune de nos propositions, les complexités associées.

Contrainte	Sémantique	Représentation	Cohérence	Filtrage
Σ-`AllDifferent`	μ^{Σ}_{var}	Réseau	O($n \times (m+n \cdot log(n))$)	O($m \times (m+n \cdot log(n))$)
Σ-`AllDifferent`	μ^{Σ}_{dec}	Conflict-set	O($e^3 \times d$)	O($m^2 \times e^3 \times d$)

TAB. 5.5 – Liste des contraintes présentées dans ce chapitre.

Chapitre 6

Relaxer Gcc et Regular avec préférences

Sommaire

Dans ce chapitre, nous présentons les relaxations des contraintes `Gcc` et `Regular` dans le cadre avec préférences. Pour la contrainte `Gcc`, nous proposons les deux sémantiques de violation suivantes : μ^{Σ}_{var} qui mesure la somme des poids des variables à ré-instancier et μ^{Σ}_{dec} qui mesure la somme pondérée des écarts aux bornes. Pour la contrainte `Regular`, nous proposons les deux sémantiques de violation suivantes : μ^{Σ}_{val} (basée sur une version pondérée de la distance de Hamming) et μ^{Σ}_{edit} (basée sur une version pondérée de la distance d'édition).

À la différence de la contrainte `AllDifferent`, pour chacune de ces quatre sémantiques, test de cohérence et filtrage associés sont de complexité polynomiale et peuvent être mis en œuvre en ajoutant aux réseaux et graphes en couches sous-jacents des arcs de violation. Toutefois, les versions relaxées de `Gcc` et de `Regular` vont jouer un rôle déterminant dans la modélisation et la résolution des `NRPs` (cf. Chapitre 8).

Nous concluons ce chapitre en montrant les liens entre ces versions relaxées et les versions « optimisation » de ces deux contraintes que sont `costGcc` [Rég02] et `costRegular` [DPR06].

6.1 Relaxation de $\texttt{Gcc}(\mathcal{X}, l, u)$ avec préférences

6.1.1 Deux sémantiques de violation avec préférences

Pour la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$, nous proposons les deux sémantiques de violation suivantes :

- la sémantique basée variables avec préférences (μ^{Σ}_{var}) qui mesure la somme des poids des variables à ré-instancier pour satisfaire la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$;
- la sémantique basée décomposition avec préférences (μ^{Σ}_{dec}) qui mesure la somme pondérée des écarts aux bornes.

Chaque variable $X_i \in \mathcal{X}$ reçoit un poids φ_i reflétant l'importance de celle-ci. La sémantique μ^{Σ}_{var} mesure la somme minimale des poids des variables qu'il est nécessaire de ré-instancier pour satisfaire la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$.

Définition 85 : ***Sémantique basée variables avec préférences pour*** $\texttt{Gcc}(\mathcal{X}, l, u)$
Soit une contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$*, telle que* $\sum_{v_j \in Doms} l_j \leq |\mathcal{X}| \leq \sum_{v_j \in Doms} u_j$*. Notons* $\mathcal{X}_{reinst}$*, l'ensemble de variables dont la somme des poids est minimale et qu'il est nécessaire de ré-instancier pour satisfaire la contrainte* $\texttt{Gcc}(\mathcal{X}, l, u)$*. La sémantique de violation basée variables avec préférences* μ^{Σ}_{var} *est définie par :*

$$\mu^{\Sigma}_{var}(\mathcal{X}) = \sum_{X_i \in \mathcal{X}_{reinst}} \varphi_i$$

La sémantique μ^{Σ}_{dec} est une généralisation de la sémantique de violation basée valeurs (voir Sous-section 4.2.1). Toute contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$ peut s'exprimer comme la conjonction de contraintes $\texttt{atleast}(\mathcal{X}, v_j, l_j)$ et $\texttt{atmost}(\mathcal{X}, v_j, u_j)$.

Définition 86 : ***Contraintes*** $\texttt{atleast}(\mathcal{X}, v_j, l_j)$ ***et*** $\texttt{atmost}(\mathcal{X}, v_j, u_j)$ ***[HSD92]***
Soit $v_j \in Doms$ *,* l_j *et* u_j *les bornes inférieures et supérieures associées à la valeur* v_j*,*

$$\begin{aligned} \texttt{atleast}(\mathcal{X}, v_j, l_j) &\equiv |\{X_i \in \mathcal{X} \mid X_i = v_j\}| \geq l_j \\ \texttt{atmost}(\mathcal{X}, v_j, u_j) &\equiv |\{X_i \in \mathcal{X} \mid X_i = v_j\}| \leq u_j \end{aligned}$$

On a l'équivalence suivante :

$$\texttt{Gcc}(\mathcal{X}, l, u) \equiv \bigwedge_{v_j \in Doms} (\texttt{atleast}(\mathcal{X}, v_j, l_j) \wedge \texttt{atmost}(\mathcal{X}, v_j, u_j))$$

Afin de pouvoir mesurer de manière fine les violations engendrées par les manques et les excès, à chaque contrainte $\texttt{atleast}(\mathcal{X}, v_j, l_j)$/$\texttt{atmost}(\mathcal{X}, v_j, u_j)$ est associée un poids $\varphi_j^{atleast}/\varphi_j^{atmost}$ représentant l'importance accordée à cette borne inférieure/supérieure. Ainsi, il est possible de distinguer les violations associées aux différentes valeurs d'une même contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$, mais aussi entre le manque et l'excès associé à une même valeur.

La sémantique μ_{dec}^{Σ} mesure, pour une instanciation d'une contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$, la somme pondérée des manques et des excès de chaque valeur.

Définition 87 : ***Sémantique basée décomposition avec préférences pour*** $\texttt{Gcc}(\mathcal{X}, l, u)$
Pour une contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$*, la sémantique de violation basée décomposition avec préférences* μ_{dec}^{Σ} *est définie par :*

$$\mu_{dec}^{\Sigma}(\mathcal{X}) = \sum_{v_j \in Doms} \left(s(\mathcal{X}, v_j) \times \varphi_j^{atleast} + e(\mathcal{X}, v_j) \times \varphi_j^{atmost} \right)$$

Nous verrons dans le chapitre 8 que cette sémantique est très pratique pour modéliser et résoudre des NRPs. En effet, elle permet de distinguer les violations associées aux différentes valeurs ainsi que leurs natures, comme le montrent les exemples ci-dessous :

- **différencier les valeurs** : les violations attachées aux équipes de nuits doivent être plus « coûteuses » que celles des autres équipes car celles-ci sont plus contraignantes pour le personnel hospitalier ;
- **différencier les natures** : le manque de repos dans le planning d'une infirmière est plus pénalisant pour celle-ci que son excès.

6.1.2 Relaxation de $\texttt{Gcc}(\mathcal{X}, l, u)$ selon la sémantique μ_{var}^{Σ}

La contrainte $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi, \mu_{var}^{\Sigma}, z)$ est la version relaxée de la contrainte globale $\texttt{Gcc}(\mathcal{X}, l, u)$ selon la sémantique de violation basée variables avec préférences μ_{var}^{Σ}.

Définition 88 : $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi, \mu_{var}^{\Sigma}, z)$
Soit c *une contrainte* $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi, \mu_{var}^{\Sigma}, z)$ *avec* $\mathcal{X}$ *une ensemble de variables,* l *et* u *les ensembles des bornes inférieures et supérieures associées aux valeurs de l'union des domaines de* $\mathcal{X}$*, tel que* $\sum_{v_j \in Doms} l_i \leq |\mathcal{X}| \leq \sum_{v_j \in Doms} u_i$*,* φ *représente le vecteur des poids associés aux*

variables de $\mathcal{X}$, et z une variable de coût, c admet une solution ssi *il existe une instanciation complète $\mathcal{A}$ telle que :*

$$\mu^{\Sigma}_{var}(\mathcal{A}) \leq max(D_z)$$

Pour illustrer le principe de cette contrainte, nous utiliserons l'exemple suivant.

Exemple 19 :
Soit la contrainte Σ-`Gcc`$(\mathcal{X}, [1,0,3], [1,1,5], [3,9,6,4], \mu^{\Sigma}_{var}, z)$ avec $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$ et les domaines $D_{X_1} = \{1\}, D_{X_2} = D_{X_3} = \{1,2\}$, $D_{X_4} = \{3\}$ et le domaine de la variable de coût $D_z = [0..10]$.

i) Arcs de violation

De la même manière que pour `soft-Gcc`$(\mathcal{X}, l, u, \mu_{var}, z)$ (voir Sous-section 4.2.2), les arcs de violation vont servir à représenter les ré-instanciations des variables de la contrainte. Or dans ce cas, chaque variable a un poids et il est nécessaire de pouvoir distinguer quelles sont les variables à ré-instancier.

Ainsi, pour chaque variable X_i et chaque valeur v_j n'appartenant pas au domaine de X_i, on ajoute au réseau de la contrainte `Gcc`$(\mathcal{X}, l, u)$ un arc de violation modélisant la réaffectation $(X_i{=}v_j)$ (non autorisée initialement) contre un coût de violation φ_i. On a ainsi l'ensemble d'arcs de violation suivant[12] :

Définition 89 : ***Arcs de violation associés à*** Σ-`Gcc`$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$

$$\tilde{A} = \{(X_i \rightarrow v_j) \mid X_i \in \mathcal{X}, v_j \notin D_{X_i}\}$$
$$avec$$
$$\forall a = (X_i \rightarrow v_j) \in \tilde{A},\ d(a) = 0,\ c(a) = 1 \text{ et } w(a) = \varphi_i$$

À chaque fois qu'une variable ne peut être instanciée sans engendrer de violation alors un arc de violation issu de celle-ci sera utilisé. Or, comme tous les arcs de violation issus d'une variable X_i ont pour poids φ_i, alors le poids minimal d'un flot faisable de valeur n dans ce réseau correspondra exactement aux poids des variables qu'il serait nécessaire de ré-instancier pour satisfaire la contrainte `Gcc`$(\mathcal{X}, l, u)$ sous-jacente.

La figure 6.1 représente le réseau associé à la contrainte Σ-`Gcc`$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$ de l'exemple 19.

ii) Test de cohérence

Par construction du réseau, le poids minimal d'un flot faisable de valeur n correspond à la somme minimale des poids des variables à ré-instancier. Donc, s'il existe un flot faisable de valeur n de poids minimal inférieur ou égal à $max(D_z)$ alors la contrainte Σ-`Gcc`$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$ est cohérente.

[12] Dans [Hoe05], le même ensemble d'arcs de violation est proposé, mais avec des poids égaux à 1.

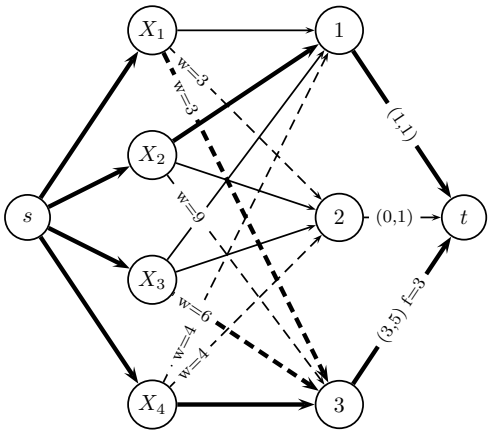


FIG. 6.1 – Réseau associé à la contrainte Σ-Gcc$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$ de l'exemple 19.

Corollaire 17 : ***Test de cohérence de*** Σ-Gcc$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$
La contrainte Σ-Gcc$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$ *admet une solution* ssi *il existe un flot faisable de valeur* n *dans le réseau associé de poids* $weight(f)$ *inférieur ou égal à* $max(D_z)$.

À nouveau, ce test de cohérence peut être effectué grâce à l'algorithme de Ford & Fulkerson en $O(n \times (m + n \cdot log(n)))$. Les arcs en gras de la figure 6.1 représentent un tel flot associé à une solution de coût 9.

iii) Filtrage

De la même façon que pour Σ-AllDifferent$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$, nous allons modifier le réseau de telle façon à pouvoir tester la viabilité d'une valeur (X_i, v_j) grâce à la recherche d'un plus court chemin reliant les sommets v_j et X_i dans le réseau résiduel.

Comme pour Σ-AllDifferent$(\mathcal{X}, W, \mu^{\Sigma}_{var}, z)$ (voir Sous-section 5.2), le fait de forcer l'utilisation d'une valeur (X_i, v_j) ne signifie pas que la variable X_i ne devra pas être ré-instanciée. Pour modéliser ce cas, il est nécessaire d'ajouter, lors du test de la valeur (X_i, v_j), des arcs de violation reliant v_j à toutes les autres valeurs.

Définition 90 : ***Arcs de violation à ajouter lors du test de viabilité de*** (X_i, v_j)

$$\tilde{A}_{X_i v_j} = \{(v_j \rightarrow v_k) \mid v_k \in Doms, v_k \neq v_j\}$$
avec
$$\forall a \in \tilde{A}_{X_i v_j},\ d(a) = 0,\ c(a) = 1,\ et\ w(a) = \varphi_i$$

Une fois le réseau résiduel ainsi augmenté, il est possible de tester la viabilité d'une valeur par la recherche de plus court chemin dans celui-ci.

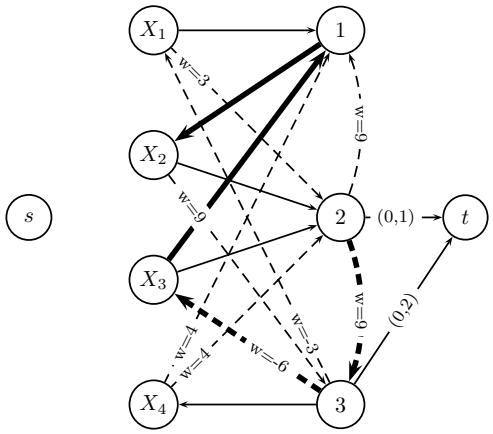


FIG. 6.2 – Réseau résiduel du flot de la figure 6.1 augmenté pour le test de cohérence de $(X_2, 2)$.

Corollaire 18 : ***Viabilité d'une valeur*** (X_i, v_j) ***pour*** Σ-Gcc$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$
Soit la contrainte Σ-Gcc$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$ *et un flot faisable* f *de valeur* n *et de poids minimal dans le réseau associé. La valeur* (X_i, v_j) *est viable* ssi *une des deux conditions suivantes est satisfaite :*

- *l'arc* $(X_i \rightarrow v_j)$ *appartient à* f *et* $weight(f) \leq max(D_z)$ *;*
- *le plus court chemin* p *reliant* v_j *et* X_i *dans le réseau résiduel de* f *augmenté des arcs de violation* $\tilde{A}_{X_i v_j}$ *a un poids* $weight(p)$ *tel que :*

$$weight(f) + weight(p) \leq max(D_z)$$

La figure 6.2 illustre, pour la valeur $(X_2, 2)$ de l'exemple 19, le réseau augmenté. Les arcs en gras représentent le plus court chemin reliant les sommets 2 et X_2. Le poids de ce plus court chemin est de 3, par conséquent la valeur $(X_2, 2)$ est incohérente $(3 + 9 > 10)$.

En utilisant le corollaire 18, il est possible d'établir la cohérence globale d'une contrainte Σ-Gcc$(\mathcal{X}, l, u, \varphi, \mu^{\Sigma}_{var}, z)$ en $O(m \times (m + n \cdot log(n)))$ par la recherche successive de m plus courts chemins.

Remarque 16 : Il est possible de réduire la complexité de l'algorithme de filtrage en ne testant que les valeurs des variables n'utilisant pas un arc de violation dans le flot initial. En effet, ces variables peuvent prendre n'importe quelle valeur de leurs domaines sans modifier le coût de l'affectation complète.

6.1.3 Relaxation de Gcc$(\mathcal{X}, l, u)$ selon la sémantique μ^{Σ}_{dec}

La contrainte Σ-Gcc$(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ est la version relaxée de la contrainte globale Gcc$(\mathcal{X}, l, u)$ selon la sémantique de violation basée décomposition avec préférences μ^{Σ}_{dec}.

Définition 91 : Σ-Gcc$(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$
Soit c *une contrainte* Σ-Gcc$(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ *avec* $\mathcal{X}$ *un ensemble de variables,*

l et u les ensembles des bornes inférieures et supérieures associées aux valeurs de l'union des domaines de $\mathcal{X}$, $\varphi^{atleast}$ (resp. φ^{atmost}) représente le vecteur des poids associés aux contraintes $\texttt{atleast}(\mathcal{X}, v_j, u_j)$ *(resp.* $\texttt{atmost}(\mathcal{X}, v_j, u_j)$*) de c et z une variable de coût, c admet une solution* ssi *il existe une instanciation complète $\mathcal{A}$ telle que :*

$$\mu^{\Sigma}_{dec}(\mathcal{A}) \leq max(D_z)$$

Pour illustrer le fonctionnement de cette contrainte, nous utiliserons l'exemple suivant :

Exemple 20 :
Soit la contrainte $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, [1,2,1], [1,3,1], [4,1,10], [2,5,1], \mu^{\Sigma}_{dec}, z)$ avec $\mathcal{X} = \{X_1, X_2, X_3, X_4\}$ ayant pour domaines $D_{X_1} = D_{X_2} = \{1\}$, $D_{X_3} = \{1,2\}$, $D_{X_4} = \{2,3\}$ et le domaine de la variable de coût $D_z = [0..10]$.

i) Arcs de violation

Comme pour $\texttt{soft-Gcc}(\mathcal{X}, l, u, \mu_{val}, z)$ (voir Sous-section 4.2.3), chaque valeur v_j possède un arc de manque $(s \rightarrow v_j)$ et un arc d'excès $(v_j \rightarrow t)$. À la différence de $\texttt{soft-Gcc}(\mathcal{X}, l, u, \mu_{val}, z)$, des poids sont associés à ces arcs pour modéliser les préférences émises. C'est pourquoi, l'arc de manque (resp. d'excès) associé à la valeur v_j reçoit le poids $\varphi^{atleast}_j$ (resp. φ^{atmost}_j).

Définition 92 : ***Arcs de violation associés à*** $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$

$$\tilde{A}_{manque} = \{(s \rightarrow v_j) \mid \forall v_j \in Doms\}$$
$$\tilde{A}_{exces} = \{(v_j \rightarrow t) \mid \forall v_j \in Doms\}$$
avec
si $a \in \tilde{A}_{manque}$ *alors* $d(a) = 0$ *et* $c(a) = l_j$ *et* $w(a) = \varphi^{atleast}_j$
si $a \in \tilde{A}_{exces}$ *alors* $d(a) = 0$ *et* $c(a) = \infty$ *et* $w(a) = \varphi^{atmost}_j$

Ainsi, à chaque fois que l'arc de manque associé à v_j est utilisé le poids du flot est augmenté de $\varphi^{atleast}_j$. De même, à chaque fois que l'arc d'excès associé à v_j est utilisé le poids du flot est augmenté de φ^{atmost}_j.

La figure 6.3 représente le réseau associé à la contrainte $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ de l'exemple 20.

ii) Maintenir la cohérence globale

Par construction, il existe une bijection entre l'ensemble des flots faisables et l'ensemble des instanciations complètes des variables de $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$. De plus, le poids d'un flot f faisable, $weight(f)$, mesure exactement le coût de violation associé à l'instanciation $\mathcal{A}$ correspondant à f (c'est-à-dire $\mu^{\Sigma}_{dec}(\mathcal{A})$).

Corollaire 19 : ***Cohérence de*** $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$
La contrainte $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ *admet une solution* ssi *il existe un flot faisable f dans le réseau associé de poids $weight(f)$ inférieur ou égal à $max(D_z)$.*

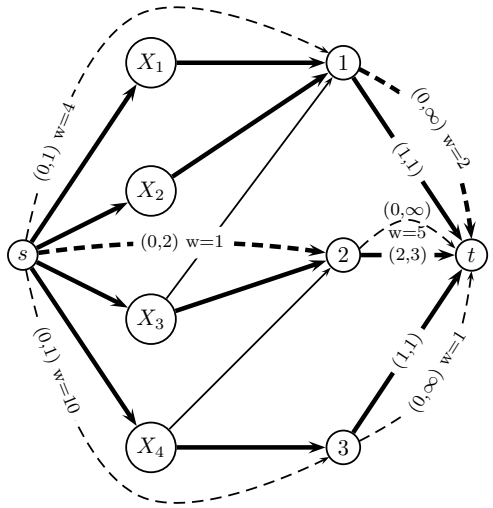


FIG. 6.3 – Réseau associé à la contrainte Σ-`Gcc`$(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ de l'exemple 20.

Comme pour `soft-Gcc`$(\mathcal{X}, l, u, \mu_{val}, z)$ (voir Sous-section 4.2.3), l'algorithme de `Ford & Fulkerson` permet de tester la cohérence globale en $O((n + \sum_{v_j \in Doms} l_j) \times (m + n \cdot log(n)))$ dans le pire des cas[13].

Les arcs en gras dans la figure 6.3 représentent un flot faisable de poids 3. Ce flot correspond à la solution suivante de coût 3 : $\{(X_1 = 1),(X_2 = 1),(X_3 = 2),(X_4 = 3)\}$.

iii) Filtrage

De la même façon que pour la sémantique de violation μ_{val} de `soft-Gcc`$(\mathcal{X}, l, u, \mu_{val}, z)$, il est possible de tester la viabilité d'une valeur (X_i, v_j) grâce à la recherche d'un plus court chemin reliant v_j à X_i.

Corollaire 20 : ***Viabilité d'une valeur*** (X_i, v_j) ***pour*** Σ-`Gcc`$(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$
Soit la contrainte Σ-`Gcc`$(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ *et un flot* f *faisable de poids minimal dans le réseau associé, une valeur* (X_i, v_j) *est viable* ssi *une des deux conditions suivantes est remplie :*

- *l'arc* $(X_i \rightarrow v_j)$ *appartient au flot* f *et* $weight(f) \leq max(D_z)$ *;*
- *il existe un chemin* p *reliant de* v_j *à* X_i *dans le réseau résiduel du flot* f *tel que :*

$$weight(f) + weight(p) \leq max(D_z)$$

La cohérence globale peut ainsi être établie en $O(min(n, k) \times (m + n \cdot log(n)))$. Pour rendre globalement cohérente la contrainte de l'exemple 20, il est nécessaire de filtrer la valeur (X_4,2).

[13]Comme dans le cas de la relaxation de `Gcc`$(\mathcal{X}, l, u)$ selon la sémantique μ_{val}, tous les flots faisables ne seront pas de valeur n (cf. Remarque 7 page 85)

6.2 Relaxation de Regular$(\mathcal{X}, \Pi)$ avec préférences

6.2.1 Sémantiques de violation

Pour la contrainte Regular$(\mathcal{X}, \Pi)$, nous proposons les deux sémantiques de violation suivantes :

- la sémantique basée valeurs avec préférences (μ^{Σ}_{val}) utilisant une version pondérée de la distance de Hamming ;
- la sémantique basée édition avec préférences (μ^{Σ}_{edit}) utilisant une version pondérée de la distance d'édition.

La sémantique μ^{Σ}_{val} mesure la violation en fonction des substitutions qu'il est nécessaire d'appliquer à un mot m_1 pour former un mot m_2 reconnu par l'automate associé à la contrainte Regular$(\mathcal{X}, \Pi)$. À chaque substitution d'un symbole a par un symbole b est associé un poids φ_{ab} reflétant le coût de celle-ci[14].

Pour mesurer la distance entre deux mots, une version pondérée de la distance de Hamming est utilisée. La distance de Hamming pondérée, notée $\mathcal{H}^w(m_1, m_2)$, mesure pour deux mots de même longueur, la somme des poids φ_{ab} des symboles qui diffèrent à une même position (le symbole a appartenant au mot m_1 et le symbole b à m_2).

Définition 93 : ***Sémantique de violation basée valeurs avec préférences pour*** Regular$(\mathcal{X}, \Pi)$ *Soit une contrainte* Regular(X, Π)*, et W une matrice contenant le coût associé aux substitutions d'un symbole par un autre, la sémantique de violation basée valeurs avec préférences μ^{Σ}_{val} est définie par :*

$$\mu^{\Sigma}_{val}(\mathcal{X}) = min\{\mathcal{H}^w(D, \mathcal{X}) \mid D = D_{X_1} \times ... \times D_{X_n} \ t.q.\ D \in L(\Pi)\}$$

Cette sémantique est une généralisation de la sémantique basée variables μ_{var} (présentée dans la sous-section 4.3.1) pour laquelle le poids associé à chaque substitution est égal à 1.

Soit un NRP et une expression régulière représentant les enchaînements d'équipes. La sémantique de violation μ^{Σ}_{val} permet de modéliser les situations suivantes : dans le planning d'un employé, une journée de repos "prévue" est substituée par une journée travaillée, la violation de cette substitution ne peut pas raisonnablement être la même si cet employé doit travailler en équipe du matin ou en équipe de nuit.

La sémantique μ^{Σ}_{edit} mesure la somme pondérée des opérations de bases (substitution, insertion, effacement) nécessaires à la transformation d'un mot m_1 en un mot m_2. À chacune de ces opérations est associée un poids modélisant l'impact de cette modification. Le poids associé à la substitution (resp. à l'insertion et à l'effacement) est noté φ^{subs} (resp. φ^{ins} et φ^{del}). Pour mesurer la violation, il est nécessaire d'utiliser une version pondérée de la distance d'édition. La distance d'édition pondérée, notée $\mathcal{E}^w(m_1, m_2)$, mesure pour deux mots la somme minimale des poids des opérations nécessaires pour transformer le mot m_1 en le mot m_2.

[14] Il est possible que φ_{ab} et φ_{ba} aient des valeurs différentes

Définition 94 : ***Sémantique de violation basée édition avec préférences pour*** `Regular`$(\mathcal{X}, \Pi)$
Soit une contrainte `Regular`(X, Π)*, et W une matrice contenant le coût associé aux différentes opérations élémentaires, la sémantique de violation basée édition avec préférences μ^{Σ}_{edit} est définie par :*

$$\mu^{\Sigma}_{edit}(\mathcal{X}) = min\{\mathcal{E}^w(D, \mathcal{X}) \mid D = D_1 \times ... \times D_n \ t.q.\ D \in L(\Pi)\}$$

On peut voir l'utilité d'une telle sémantique pour corriger le texte écrit par un opérateur. Dans certains cas, on peut constater que certaines des opérations élémentaires, transformant un mot en un autre, ne sont pas toutes aussi probables les unes que les autres.

6.2.2 Relaxation de `Regular`$(\mathcal{X}, \Pi)$ selon la sémantique μ^{Σ}_{val}

Pour relaxer la contrainte `Regular`$(\mathcal{X}, \Pi)$ selon la sémantique basée valeurs avec préférences μ^{Σ}_{val}, nous proposons la contrainte Σ-`Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$.

Définition 95 : Σ-`Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$
Soit c une contrainte Σ-`Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$ *avec $\mathcal{X}$ un ensemble de variables, Π un automate fini déterministe, W une matrice contenant le coût des substitutions de tous les symboles de l'alphabet de Π et z une variable de coût. La contrainte c admet une solution* ssi *il existe une instanciation complète $\mathcal{A}$ telle que :*

$$\mu^{\Sigma}_{val}(\mathcal{A}) \leq max(D_z)$$

Dans la suite de cette section, nous utiliserons l'exemple suivant.

Exemple 21 :
Soit la contrainte Σ-`Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$ avec $\mathcal{X} = \{X_1, X_2, X_3, X_4, X_5\}$ ayant pour domaines $D_{X_1} = D_{X_2} = D_{X_3} = \{a, b, c\}$, $D_{X_4} = \{a, b\}$, $D_{X_5} = \{b, c\}$ et $D_{X_z} = [0..4]$. La matrice de substitutions W et l'automate Π sont tels que décrits par les figures 6.4 et 6.5.

m_1 \ m_2	a	b	c
a	–	1	4
b	3	–	2
c	5	9	–

FIG. 6.4 – Matrice des substitutions W.

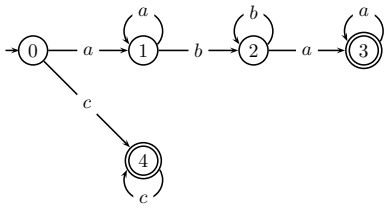


FIG. 6.5 – Automate Π.

i) Arcs de violation

À chaque transition $\delta(q_l, v_j) = q_m$ de l'automate sortant de la couche V_{i-1}, on ajoute un arc de violation pour chaque valeur $v_k \in D_{X_i} \backslash \{v_j\}$ modélisant le fait d'utiliser v_k plutôt que v_j. Cet arc de violation est étiqueté par la valeur v_k et a pour poids φ_{v_j, v_k}.

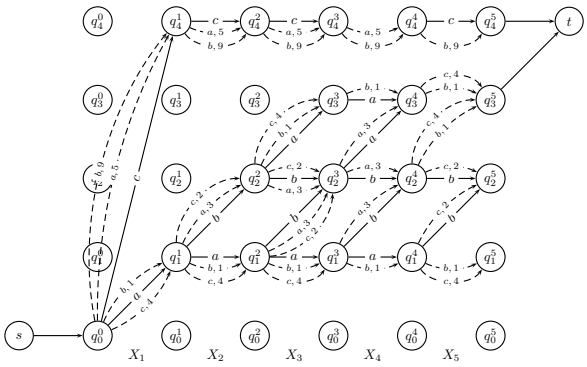


FIG. 6.6 – Graphe en couches associé à la contrainte Σ-Regular$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$ de l'exemple 21 après la phase ascendante.

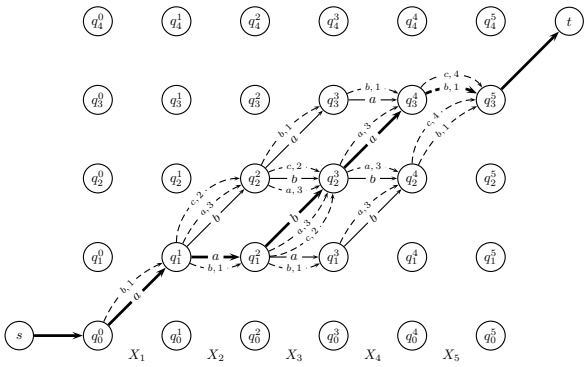


FIG. 6.7 – Graphe en couches associé à la contrainte Σ-Regular$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$ de l'exemple 21 après la phase descendante.

Définition 96 : ***Arcs de violation (substitution) associés à*** Σ-Regular$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$

$$A^{\Sigma}_{subs} = \{(q_l^{i-1} \rightarrow q_m^i) \mid \delta(q_l, v_j) = q_m,\ v_k \in D_{X_i} \backslash \{v_j\},\ i = 1, ..., n\}$$
Soit a un arc représentant la substitution de v_j par v_k
alors a est étiqueté par v_k et $w(a) = \varphi_{v_j, v_k}$

Chaque fois qu'est utilisé un arc de violation représentant une substitution du symbole v_j par v_k alors le poids du plus court chemin passant par cet arc est augmenté de φ_{v_j, v_k}.

Les figures 6.6 et 6.7 représentent respectivement la construction du graphe en couches à la fin des phases ascendante et descendante.

ii) Test de cohérence

Comme pour la contrainte soft-Regular$(\mathcal{X}, \Pi, \mu_{var}, z)$ (voir Sous-section 4.3.2), le graphe en couches est acyclique. Et par construction, tout chemin de s à t de poids inférieur ou égal à $max(D_z)$ correspond à une solution.

Corollaire 21 : ***Cohérence de*** Σ-Regular$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$
La contrainte Σ-Regular$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$ *admet une solution* ssi *il existe un s–t chemin p dans le graphe en couches associé de poids $weight(p)$ inférieur ou égal à $max(D_z)$.*

Le test de cohérence peut ainsi être réalisé grâce à un parcours en largeur du graphe en couches en $O(n \times |Q| \times |\Sigma|)$ dans le pire cas.

Les arcs en gras dans la figure 6.7 représentent le chemin $\{s, q_0^0, a, q_1^1, a, q_1^2, b, q_2^3, a, q_3^4, b, q_3^5, t\}$ de poids 1 correspondant à la solution $\{(X_1 = a), (X_2 = a), (X_3 = b), (X_4 = a), (X_5 = b)\}$ de coût 1.

iii) Filtrage

De même que pour `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$ on retrouve la caractérisation suivante pour tester la viabilité d'une valeur.

Corollaire 22 : ***Viabilité de*** (X_i, v_j) ***pour*** Σ`-Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$
Soit une contrainte Σ`-Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$*, la valeur* (X_i, v_j) *est viable* ssi *il existe au moins un chemin* p *reliant* s *à* t *dans le graphe en couches associé de poids inférieur ou égal à* $max(D_z)$ *et utilisant un arc étiqueté par* v_j *reliant les couches* V_{i-1} *et* V_i*.*

Le même algorithme que pour `soft-Regular`$(\mathcal{X}, \Pi, \mu_{var}, z)$ peut être ainsi utilisé. Ainsi, la cohérence globale peut être établie en $O(n \times |Q| \times |\Sigma|)$ dans le pire cas (voir Sous-section 4.3.2).

L'exemple 21 peut être rendu globalement cohérent en filtrant la valeur (X_1, c).

6.2.3 Relaxation de `Regular`$(\mathcal{X}, \Pi)$ selon la sémantique μ^{Σ}_{edit}

La contrainte Σ`-Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{edit}, z)$ correspond à la relaxation de `Regular`$(\mathcal{X}, \Pi)$ selon la sémantique basée édition avec préférences μ^{Σ}_{edit}.

Définition 97 : Σ`-Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{edit}, z)$
Soit c *une contrainte* Σ`-Regular`$(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{edit}, z)$ *avec* $\mathcal{X}$ *un ensemble de variables,* Π *un automate fini déterministe,* W *une matrice contenant les coûts associés à chaque opération élémentaire et* z *une variable de coût. La contrainte* c *admet une solution* ssi *il existe une instanciation complète* $\mathcal{A}$ *telle que*

$$\mu^{\Sigma}_{edit}(\mathcal{A}) \leq max(D_z)$$

i) Arcs de violation

L'ensemble d'arcs de violation reste identique à celui de la sémantique μ_{edit} (voir Sous-section 4.3.3) car les opérations élémentaires ne sont pas modifiées. En revanche, ces arcs ne sont plus tous de poids identiques mais sont pondérés par les poids associés aux opérations correspondantes.

Ainsi, un arc de violation représentant une insertion recevra pour poids φ^{ins}.

ii) Test de cohérence et filtrage

Le graphe en couches de la version avec préférences possède exactement la même structure que le cas sans préférences. Il est possible d'appliquer immédiatement le même test de cohérence et le même algorithme de filtrage que pour `soft-Regular`$(\mathcal{X}, \Pi, \mu_{edit}, z)$ (voir Sous-section 4.3.3).

6.3 Liens entre les versions relaxations et les versions optimisations

Dans cette section, nous montrons que les versions optimisations de $\texttt{Gcc}(\mathcal{X}, l, u)$ et de $\texttt{Regular}(\mathcal{X}, \Pi)$, que sont $\texttt{costGcc}(\mathcal{X})$ [Rég02] et $\texttt{costRegular}(\mathcal{X}, \Pi, z)$ [DPR06], permettent de formuler la relaxation de $\texttt{Gcc}(\mathcal{X}, l, u)$ et de $\texttt{Regular}(\mathcal{X}, \Pi)$ selon certaines sémantiques de violation.

6.3.1 Σ-$\texttt{Regular}(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$ vs. $\texttt{costRegular}(\mathcal{X}, \Pi', z)$

Toute contrainte Σ-$\texttt{Regular}(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$ peut se formuler à l'aide d'une contrainte $\texttt{costRegular}(\mathcal{X}, \Pi', z)$, où Π' est l'automate Π auquel on a ajouté de nouvelles transitions (avec poids). Ces transitions vont permettre de modéliser et quantifier la violation. Les deux contraintes étant représentées par le même graphe en couches, tests de cohérence et filtrages sont analogues.

i) La contrainte $\texttt{costRegular}(\mathcal{X}, \Pi, z)$

Définition 98 : $\texttt{costRegular}(\mathcal{X}, \Pi, z)$ ***[DPR06]***
Soit z une variable objectif et Π un automate dont les transitions sont pondérées, la contrainte $\texttt{costRegular}(\mathcal{X}, \Pi, z)$ admet une solution ssi *il existe une instanciation complète $\mathcal{A}$ telle que la contrainte $\texttt{Regular}(\mathcal{X}, \Pi)$ est cohérente et que la somme des coûts des transitions utilisées par $\mathcal{A}$ soit inférieure ou égale à $max(D_z)$.*

Le graphe en couches de $\texttt{costRegular}(\mathcal{X}, \Pi, z)$ est obtenu de la même manière que pour $\texttt{Regular}(\mathcal{X}, \Pi)$, mais en ajoutant aux arcs des poids correspondant à ceux des transitions associées [DPR06]. L'existence d'un s–t chemin de poids inférieur ou égal à $max(D_z)$ permet de tester la cohérence de la contrainte.

ii) Exprimer Σ-$\texttt{Regular}(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$ avec un $\texttt{costRegular}(\mathcal{X}, \Pi', z)$

Soit l'automate Π associé à la contrainte Σ-$\texttt{Regular}(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$. Pour exprimer la relaxation de $\texttt{Regular}(\mathcal{X}, \Pi)$ (selon la sémantique de violation μ^{Σ}_{val}) à l'aide de la contrainte $\texttt{costRegular}(\mathcal{X}, \Pi, z)$, il est nécessaire d'ajouter à l'automate initial Π des transitions représentant les différentes substitutions possibles.

Pour chaque transition $\delta(q_l, v_j) = q_m$ de Π et pour chaque symbole v_k de $\Sigma \backslash \{v_j\}$ (Σ étant l'alphabet utilisé par Π), on ajoute une nouvelle transition $\delta(q_l, v_k) = q_m$ de poids égal à φ_{v_j, v_k}. Nous noterons Π' ce nouvel automate.

La contrainte $\texttt{costRegular}(\mathcal{X}, \Pi', z)$ est une relaxation de la contrainte $\texttt{Regular}(\mathcal{X}, \Pi)$ selon la sémantique de violation μ^{Σ}_{val}.

Exemple 22 :
Soit la contrainte Σ-$\texttt{Regular}(\{X_1, X_2, X_3\}, \Pi, W, \mu^{\Sigma}_{val}, z)$ avec $D_{X_1} = D_{X_2} = D_{X_3} = \{a, b, c\}$

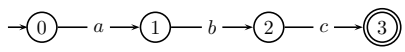


FIG. 6.8 – Automate Π.

m_1 \ m_2	a	b	c
a	–	1	2
b	3	–	4
c	5	6	–

FIG. 6.9 – Matrice des coûts W.

et $D_z = [0..10]$. Les figures 6.8 et 6.9 décrivent respectivement l'automate Π et la matrice des coûts W associés aux substitutions.

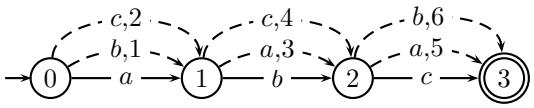


FIG. 6.10 – Automate Π′.

La figure 6.10 décrit l'automate Π′ associé à la relaxation de la contrainte $\texttt{Regular}(\mathcal{X}, \Pi)$ de l'exemple 22.

Le nouvel automate peut ne pas être déterministe. Toutefois, cela n'empêche pas son déploiement en un graphe en couches [QW06]. De plus, comme la complexité de la contrainte $\texttt{costRegular}(\mathcal{X}, \Pi, z)$ est directement liée au nombre d'états de l'automate associé, il n'est pas conseillé de déterminiser Π′ (la déterminisation de celui peut ajouter un nombre important d'états [RS59]).

La figure 6.11 représente le graphe en couches obtenu par le déploiement de l'automate Π′. Ce graphe en couches est identique à celui de la contrainte $\Sigma\text{-}\texttt{Regular}(\mathcal{X}, \Pi, W, \mu^{\Sigma}_{val}, z)$.

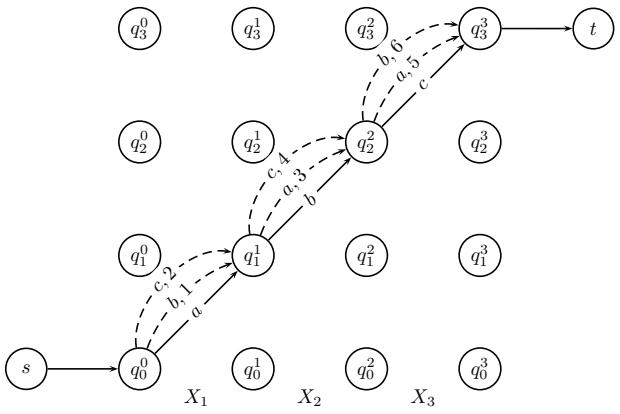


FIG. 6.11 – Graphe en couches déployé à partir de l'automate Π′.

La sémantique de violation μ^{Σ}_{val} étant une généralisation de la sémantique μ_{var}, cette transformation permet aussi de modéliser la relaxation de la contrainte $\texttt{Regular}(\mathcal{X}, \Pi)$ selon la sémantique de violation basée variables (μ_{var}).

iii) Remarques

Comme $\texttt{costRegular}(\mathcal{X}, \Pi, z)$ permet d'exprimer la relaxation de $\texttt{Regular}(\mathcal{X}, \Pi)$ selon la sémantique μ^{Σ}_{var}, nous utiliserons cette contrainte dans le chapitre 8.

6.3.2 Σ-$\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ vs. $\texttt{costGcc}(\mathcal{X}', W, l', u', z)$

Toute contrainte Σ-$\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ peut se formuler à l'aide de la contrainte $\texttt{costGcc}(\mathcal{X}', W, l', u', z)$. Mais cette transformation nécessite de dupliquer les valeurs, d'introduire de nouvelles variables et d'ajouter des arcs au réseau de $\texttt{costGcc}(\mathcal{X}, W, l, u)$. Si d'un point de vu formel cette transformation est valide, l'augmentation de la taille du réseau interdit son utilisation pratique.

i) La contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$

Définition 99 : $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$ ***[Rég02]***
Soit z une variable objectif et W une matrice de coûts associés aux affectations des variables de $\mathcal{X}$. La contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$ *admet une solution* ssi *il existe une instanciation complète $\mathcal{A}$ telle que la contrainte* $\texttt{Gcc}(\mathcal{X}, l, u)$ *soit cohérente et que la somme des coûts des affectations de $\mathcal{A}$ soit inférieure ou égale à $max(D_z)$.*

Les arcs issus du graphe de valeurs (du réseau de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$) sont pondérés de tel sorte à modéliser les poids de W [Rég02]. Le test de cohérence consiste maintenant en la recherche d'un flot faisable de valeur n de poids inférieur ou égal à la valeur maximale d'une variable objectif z. La contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$ peut être rendue globalement cohérente en $O(n \times (m + n \cdot log(n)))$ [Rég02].

Pour illustrer cette transformation, nous utiliserons l'exemple suivant.

Exemple 23 :
Soit la contrainte Σ-$\texttt{Gcc}(\{X_1, X_2, X_3\}$,[1,2][1,2],{3,5}{4,2},$z)$ avec $D_{X_1} = D_{X_2} = \{1\}$, $D_{X_3} = \{2\}$ et D_z=[0..9]

La figure 6.12 représente le réseau construit pour Σ-$\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$, et les arcs en gras un flot faisable de poids minimal associé à la solution $\{(X_1 = 1), (X_2 = 1), (X_3 = 2)\}$ de coût 9.

ii) Représenter l'excès

Afin de pouvoir modéliser les excès dans le réseau d'une contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$, il est nécessaire d'ajouter des sommets et des arcs pour « re-router » le flot excédentaire.

Pour chaque valeur v_j est ajouté un sommet v_je. Ces sommets sont reliés au puits par un arc de poids nul (comme les valeurs « classiques » dans $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$) avec une demande

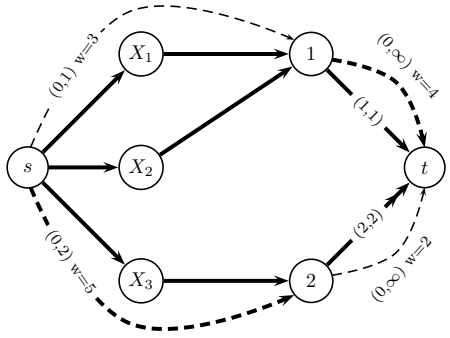


FIG. 6.12 – Réseau associé à la contrainte Σ-`Gcc`$(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, z)$ de l'exemple 23.

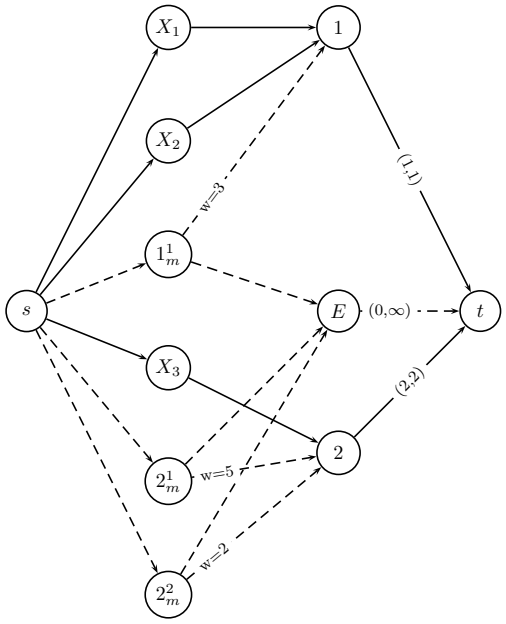


FIG. 6.13 – Ajout des arcs et sommets modélisant le manque dans le réseau.

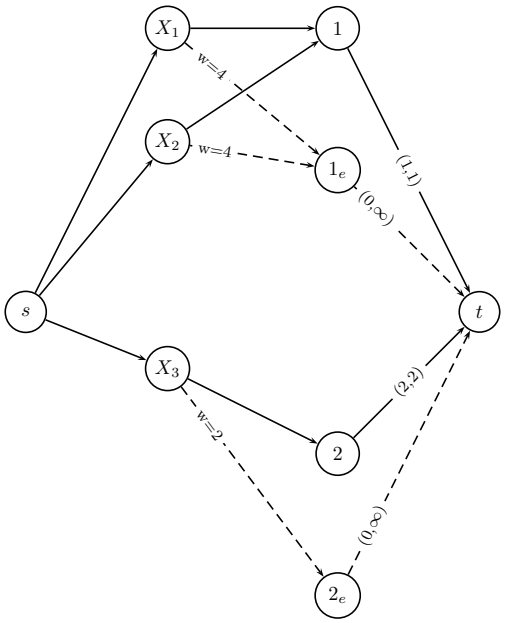


FIG. 6.14 – Ajout des arcs et sommets modélisant l'excès dans le réseau.

nulle et une capacité infinie. Chaque variable X_i possédant dans son domaine la valeur v_j est connectée au sommet v_je, exactement comme si v_je appartenait au domaine de X_i. Ces arcs ont pour demande et capacité respective 0 et 1, et sont de poids φ_j^{atmost}. Ainsi, si la variable X_i génère un excès alors l'arc $(X_i \rightarrow v_je)$ est utilisé. Comme cet arc est de poids φ_j^{atmost} alors le poids du flot est augmenté de φ_j^{atmost}. La figure 6.14 représente le réseau de la contrainte `costGcc`$(\mathcal{X}, W, l, u, z)$ après l'ajout des sommets et des arcs servant à modéliser l'excès.

iii) Représenter le manque

Il est possible de modéliser le manque par l'ajout de nouvelles variables permettant de « combler » les demandes de chacune des valeurs.

Pour chaque valeur v_j, sont ajoutées l_j nouvelles variables (elles sont notées $v_j{}^k_m$ pour $1 \leq$

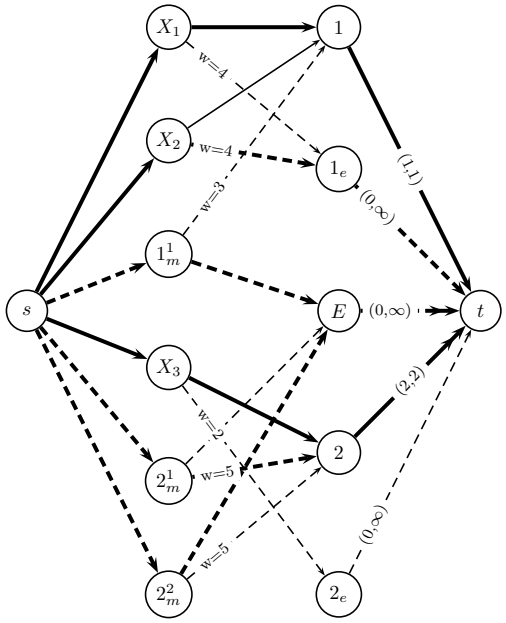


FIG. 6.15 – Réseau de la contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$ permettant de modéliser la contrainte de l'exemple 23.

$k \leq l_j$). Ces nouvelles variables sont connectées à la source comme les variables « classiques » (c'est-à-dire par un arc de demande 1 et de capacité 1) et sont reliées aux valeurs qu'elles peuvent « combler ». L'arc $(v_j{}^k_m \rightarrow v_j)$ a donc une demande nulle une capacité de 1 et est de poids $\varphi_j^{atleast}$. Ainsi, si cet arc est utilisé (pour satisfaire la borne inférieure) alors le poids du flot sera augmenté de $\varphi_j^{atleast}$.

Ces variables n'ont pas toujours besoin d'être utilisées pour atteindre les bornes inférieures de chaque valeur. C'est pourquoi, il est nécessaire d'ajouter une valeur E servant à « re-router » le flot excédentaires provenant de ces variables $v_j{}^k_m$ de manière à ce qu'il puisse exister un flot faisable. La figure 6.13 représente le réseau de la contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$ après l'ajout des sommets et des arcs servant à modéliser le manque.

iv) Remarques

Après cette transformation le test de cohérence de la contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$ peut être directement appliqué. La figure 6.15 représente le réseau d'une contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$ équivalente à la contrainte $\Sigma\texttt{-Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ de l'exemple 23. Les arcs en gras représentent le flot associé à la solution $\{(X_1 = 1),(X_2 = 1),(X_3 = 2)\}$ de coût 9.

La complexité temporelle des algorithmes de flot est liée à la taille du réseau. L'utilisation d'une contrainte $\texttt{costGcc}(X, W, l, u, z)$ exprimant la relaxation de $\texttt{Gcc}(\mathcal{X}, l, u)$ selon la sémantique μ^{Σ}_{dec} nécessite d'ajouter un nombre important d'arcs et de sommets, ce qui n'est pas le cas pour la contrainte $\Sigma\texttt{-Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$. En effet pour exprimer cette relaxation à l'aide d'une contrainte $\texttt{costGcc}(\mathcal{X}, W, l, u, z)$, il est nécessaire d'ajouter $(\sum_{v_j \in Doms} l_j +$

$k+1$) sommets et ($3 \times \sum_{v_j \in Doms} l_j + m + k + 1$) arcs au réseau de la contrainte $\texttt{Gcc}(\mathcal{X}, l, u)$. L'utilisation d'une contrainte $\Sigma\text{-}\texttt{Gcc}(\mathcal{X}, l, u, \varphi^{atleast}, \varphi^{atmost}, \mu^{\Sigma}_{dec}, z)$ ne nécessite l'ajout que de $2 \times k$ arcs.

Cette transformation peut aussi être utilisée pour modéliser la version sans préférences de la sémantique basée décomposition. Pour cela, il suffit de fixer les poids non nuls à 1.

6.4 Conclusion

Dans ce chapitre, nous avons présenté différentes sémantiques de violation permettant l'expression de préférences pour les contraintes $\texttt{Gcc}(\mathcal{X}, l, u)$ et $\texttt{Regular}(\mathcal{X}, \Pi)$ permettant l'expression de préférences. Le tableau 6.1 résume les sémantiques de violation que nous avons proposées.

Contrainte	Mesure	Sémantique
`Gcc`	Somme minimale des poids des variables à ré-instancier	μ^{Σ}_{var}
	Somme pondérée des manques et des excès	μ^{Σ}_{dec}
`Regular`	Distance de Hamming pondérée	μ^{Σ}_{val}
	Distance d'édition pondérée	μ^{Σ}_{edit}

TAB. 6.1 – Tableau récapitulant pour chaque contrainte les différentes sémantiques proposées.

Les différentes relaxations que nous avons proposées reposent sur l'ajout d'arcs de violation. Les tests de cohérence et algorithmes de filtrages induits sont de complexité temporelle similaire à celles des versions sans préférences (Tableau 6.2).

Contrainte	Sémantique	Représentation	Cohérence	Filtrage
Σ-`Gcc`	μ^{Σ}_{var}	Réseau	$O(n \times (m+n \cdot log(n)))$	$O(m \times (m+n \cdot log(n)))$
Σ-`Gcc`	μ^{Σ}_{dec}	Réseau	$O((n+\sum_{v_j \in Doms} l_j) \times (m+n \cdot log(n)))$	$O(min(n,k) \times (m+n \cdot log(n)))$
Σ-`Regular`	μ^{Σ}_{val}	Graphe en couches	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$
Σ-`Regular`	μ^{Σ}_{edit}	Graphe en couches	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$	$O(n \times \lvert Q \rvert \times \lvert \Sigma \rvert)$

TAB. 6.2 – Liste des contraintes présentées dans ce chapitre.

Nous verrons dans le chapitre 8 que ces nouvelles contraintes permettant l'utilisation de mesures de violation fines autorisant l'expression de préférences sont très utiles, notamment pour les NRPs.

Chapitre 7

Problème de la communication entre contraintes globales

Sommaire

Le filtrage des contraintes globales (relaxées ou non) permet de maintenir des cohérences fortes de manière locale à chacune de ces contraintes. Lors de la résolution d'un problème où interviennent plusieurs contraintes globales, leurs filtrages communiquent entre eux uniquement au fil de la réduction des domaines de leurs variables partagées. En outre, les contraintes globales relaxées communiquent aussi au travers des domaines de leurs variables objectifs. Ce mode de communication peut parfois s'avérer assez faible et ne pas détecter des valeurs non-viables qu'un filtrage plus global aurait décelées.

Deux approches principales permettent d'améliorer la communication dans le cadre des réseaux de contraintes classiques ou valués :

- Le transport de coûts, où des opérateurs permettent de déplacer les coûts entre les contraintes d'un réseau,
- La définition de nouvelles contraintes (globales) agglomérant plusieurs contraintes élémentaires ou globales.

Dans ce chapitre, nous motivons et décrivons chacune de ces deux approches et nous présentons nos travaux permettant d'agglomérer une contrainte `Regular` avec une contrainte de cardinalité (`atmost` et/ou `atleast`). Nous montrerons l'intérêt et les apports d'une telle contrainte (et de sa version relaxée) sur un ensemble de problèmes d'affectation de personnels.

7.1 Transport de coûts dans les VCSPs additifs

7.1.1 Cas des réseaux de contraintes binaires

Comme nous l'avons indiqué dans le chapitre 3, le renforcement de cohérence dans les Weighted CSPs se fait par l'application répétée d'opérations « atomiques » permettant de transférer les coûts entre contraintes. Ces opérations transforment le problème initial en un problème équivalent :

Définition 100 : ***Équivalence entre Weighted CSPs [LS03]***
Soient P et P' deux Weighted CSPs portant sur le même ensemble de variables. P et P' sont dits équivalents ssi *ils définissent la même distribution de coûts sur les affectations complètes.*

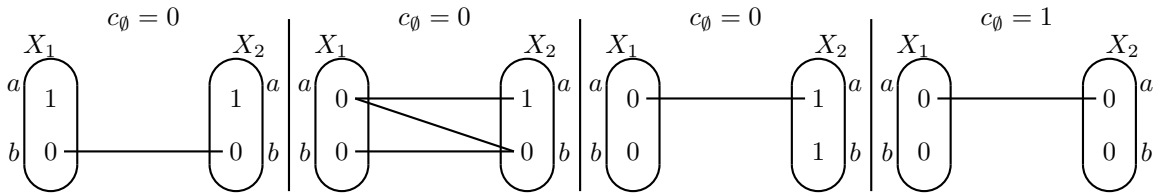
Le calcul du minorant repose sur l'existence de la contrainte d'arité zéro, noté $c_\emptyset$, qui représente un minorant global du problème (c'est-à-dire de toutes affectations complètes).

Remarque 17 : En plus de cette contrainte $c_\emptyset$, nous admettrons l'existence d'une contrainte unaire c_{X_i} pour chacune des variables du problème.

Le transport de coûts s'effectue par le biais de deux opérations :

- le transfert du coût d'une contrainte binaire vers une contrainte unaire (et inversement) ;
- le transfert du coût d'une contrainte unaire vers la contrainte d'arité zéro $c_\emptyset$.

L'exemple 24 permet d'illustrer ces différentes opérations.



(a) Réseau P (b) Réseau P' (c) Réseau P'' (d) Réseau P'''

FIG. 7.1 – Transport de coûts dans le réseau de contraintes de l'exemple 24.

Exemple 24 :
Soit P un Weighted CSP contenant deux variables ayant pour domaine {a,b} et quatre contraintes : une contrainte binaire $c_{X_1X_2}$, deux contraintes unaires c_{X_1} et c_{X_2} et la contrainte $c_\emptyset$.

La figure 7.1 représente quatre Weighted CSPs équivalents. Les valuations des contraintes unaires sont représentées par les nombres inscrits à l'intérieur des variables. Les arêtes entre deux variables représentent les tuples interdits (si aucun poids ne figure sur un arc alors le poids de celui-ci est égal à 1).

Pour transformer le réseau de contraintes P en P', il est nécessaire de projeter le coût de la contrainte unaire $(X_1 = a)$ sur la contrainte binaire. Pour cela la valuation de $(X_1 = a)$ est diminuée de 1 et les valuations de tous les tuples de la contrainte binaire $c_{X_1X_2}$, utilisant la valeur (X_1, a), sont augmentés de 1.

Pour obtenir P'' à partir de P', il faut transférer les coûts des tuples de la contrainte binaire utilisant la valeur (X_2, b) sur la contrainte unaire c_{X_2}. Les tuples (a, b) et (b, b) voient leur valuation diminuée de 1 et la valuation de $(X_2 = b)$ est augmentée de 1.

Finalement, dans le réseau de contraintes P'', toutes les valeurs de la variable X_2 engendrent un coût de 1. Il est alors possible de déplacer un coût 1 de la contrainte unaire c_{X_2} vers la contrainte $c_\emptyset$.

Depuis le début des années 2000, différentes cohérences locales ont été proposées pour les Weighted CSPs. Celles-ci affinent les notions définies pour les CSPs en utilisant ces mécanismes de transport de coûts. Dans ce qui suit, la notation $c_{X_i}(a)$ (resp. $c_{X_iX_j}(a, b)$) désigne la valuation de la contrainte c_{X_i} pour la valeur a (resp. de la contrainte $c_{X_iX_j}$ pour le tuple (a, b)).

La *cohérence de nœud* (`NC*`, [Lar02]) : une valeur (X_i, a) est dite nœud-cohérente *ssi* $c_{X_i}(a)+c_\emptyset<k$, avec k étant la valuation de la meilleure solution connue (ou à $\top$ si aucune solution n'est connue). Une variable X_i est nœud-cohérente *ssi* toutes ses valeurs sont nœud-cohérentes. Un Weighted CSP est nœud-cohérent *ssi* toutes ses variables le sont.

Définition 101 : ***Notion de support pour les contraintes binaires***
Soient $c_{X_iX_j}$ une contrainte binaire et (X_i, a) une valeur. La valeur (X_j, b) est un support de (X_i, a) pour $c_{X_iX_j}$ ssi $c_{X_iX_j}(a, b)$=0.

La *cohérence d'arc* (AC*, [Sch00]) : une valeur (X_i, a) est arc-cohérente *ssi* elle est nœud-cohérente et que pour toute contrainte binaire $c_{X_iX_j}$ il existe au moins un support pour celle-ci. Une variable est arc-cohérente *ssi* toutes ses valeurs le sont. Un Weighted CSP est AC* si toutes ses variables sont AC*.

Définition 102 : ***Notion de support complet pour les contraintes binaires***
Soient c_{X_i,X_j} une contrainte binaire et une valeur (X_i, a). La valeur (X_j, b) est un support complet de (X_i, a) pour $c_{X_iX_j}$ ssi *$c_{X_iX_j}(a, b) \oplus c_{X_j}(b)$=0.*

La *cohérence d'arc directionnelle* (DAC*, [Coo03, LS03]) : étant donné un ordre sur les variables, DAC* teste si chaque variable est nœud-cohérente et si pour toute contrainte $c_{X_iX_j}$, telle que $X_i < X_j$, chaque valeur du domaine de X_i possède un support complet dans X_j.

La *cohérence d'arc directionnelle complète* (FDAC*, [Coo03, LS03]) : étant donné un ordre sur les variables, FDAC* teste pour chaque variable si celle-ci est AC* et DAC*.

D'autres cohérences plus fortes ont été proposées, notamment :

- EAC* [dGHZL05] qui teste pour chaque variable X_i s'il existe au moins une valeur (X_i, a) ayant un support complet pour toute contrainte binaire $c_{X_iX_j}$;
- EDAC* [dGHZL05] qui combine FDAC* avec EAC* ;
- OSAC [CdGS07] qui permet de déterminer un ensemble de projections permettant de maximiser la valuation de $c_\emptyset$;
- VAC [CdGS$^+$08] qui utilise une version « satisfaction » du problème initial pour détecter les ensembles de valeurs filtrées provoquant les incohérences du problème.

7.1.2 Extensions de AC* aux réseaux contenant des contraintes globales relaxées

Malgré leur puissance, les techniques de filtrage des Weighted CSPs ne s'appliquent pas (ou très mal) aux contraintes globales car elles se restreignent uniquement aux réseaux de contraintes unaires, binaires et ternaires. Les contraintes portant sur plus de trois variables doivent d'abord être converties en leur version binaire ou être activées dès que suffisamment de variables ont été instanciées pendant la recherche.

En 2009, Jimmy H.M. Lee et Ka Lun Leung ont montré pour la première fois que les techniques de filtrage des Weighted CSPs peuvent être combinées avec les mécanismes de filtrage des contraintes globales relaxées [LL09]. Une version généralisée de AC* (GAC*) est aussi proposée, celle-ci permet de traiter certaines contraintes globales en temps polynomial.

Soit c une contrainte n-aire et X_i une variable. Une variable X_i est dite globalement cohérente (GAC*, [CS04, LL09]) *ssi* elle est nœud-cohérente (NC*) et si pour chaque valeur $v_j \in D_{X_i}$ il existe une valeur $v_l \in D_{X_k}$ (pour tout $k \neq i$) formant un tuple valide pour c. Un Weighted CSP est GAC* *ssi* toutes ses variables le sont.

Remarque 18 : GAC* est équivalente à AC* si le réseau de contraintes est constitué seulement de contraintes unaires et binaires.

Algorithme 13 : Algorithme maintenant `GAC*` pour les Weighted CSPs.

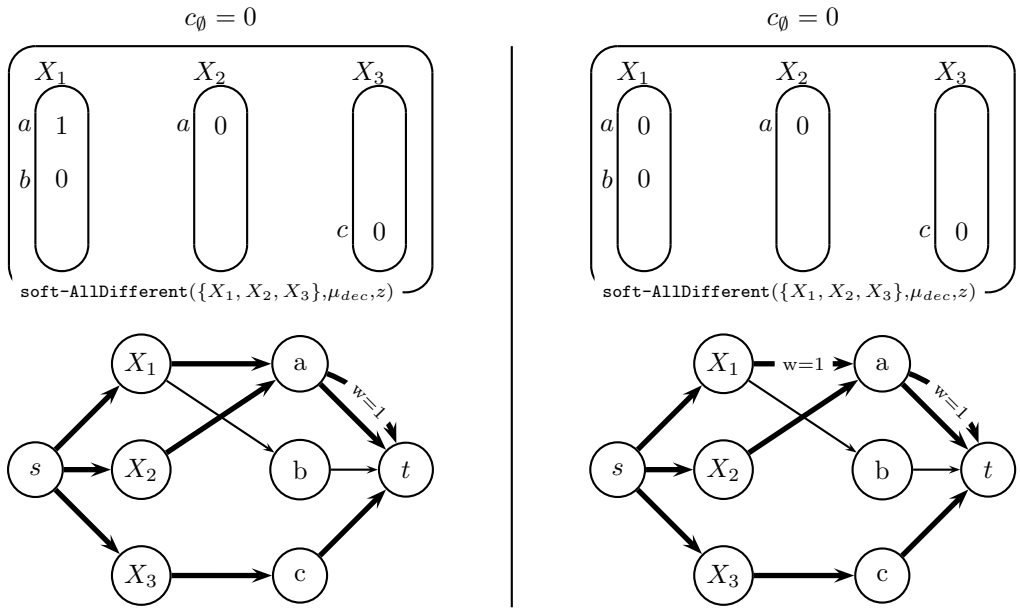
```
1  procédure enforceGAC*(𝒳 : l'ensemble de variables, 𝒞 : l'ensemble des contraintes)
2  début
3      Soit Q une file
4      Q ← 𝒳
5      tant que Q n'est pas vide faire
6          X_u ← Q.defiler()
7          flag ← false
8          pour chaque c ∈ 𝒞 telle que X_u ∈ scope(c) faire
9              pour chaque X_i ∈ scope(c)\{X_u} faire
10                 flag ← flag∨ findSupport(c_i, X_i)
11                 si pruneVal(X_i) alors Q.enfiler(X_i)
12         si flag alors
13             pour chaque X_i ∈ 𝒳 faire
14                 si pruneVal(X_i) alors Q.enfiler(X_i)
15 fin
16 fonction findSupport(c : une contrainte, X_i : une variable) : booléen
17 début
18     flag ← false
19     pour chaque v_j ∈ D_{X_i} faire
20         α ← min{c(l) | X_i = v_j}
21         si c_{X_i}(v_j) = 0 ∧ α > 0 alors flag ← true
22         c_{X_i}(v_j) ← c_{X_i}(v_j) ⊕ α
23         pour chaque tuple l avec X_i = v_j faire
24             c(l) ← c(l) ⊖ α
25     unaryProject(X_i)
26     retourner flag
27 fin
```

L'algorithme 13 permet de maintenir la cohérence globale pour un Weighted CSP. Cet algorithme a une complexité temporelle exponentielle. En effet, les lignes 20 et 23 nécessitent un parcours de la table de la contrainte globale de taille potentiellement exponentielle (voir Sous-section 3.3).

Le temps consommé par ces opérations (recherche de la valuation minimale et mise à jour des poids de tous les tuples correspondant à $\{(X_i = v_j)\}$ lors d'un transfert) peut être réduit à des temps polynomiaux dans le cas des contraintes globales relaxées dites « *projection-safe* ».

Définition 103 : ***Contrainte globale relaxée « projection-safe »***
Une contrainte globale relaxée c avec une sémantique de violation μ est dite projection-safe ssi *les trois conditions suivantes sont remplies :*

- *la relaxation de la contrainte globale c selon la sémantique μ peut être représentée par un réseau R ;*
- *il existe une bijection entre les flots dans R et les instanciations complètes des variables*



(a) Avant transfert du coût de la contrainte unaire $c_{X_1}(a)$

(b) Après transfert du coût de la contrainte unaire $c_{X_1}(a)$

FIG. 7.2 – Exemple du transfert d'un coût d'une contrainte unaire vers une contrainte globale.

de c ;

- *soit X_i une variable du scope de la contrainte c, si $X_i = a$ alors une unité de flot doit passée par l'arc ($X_i \rightarrow a$) et sinon aucune unité de flot ne doit circulée par ($X_i \rightarrow a$).*

Les contraintes suivantes sont « projection-safe » :

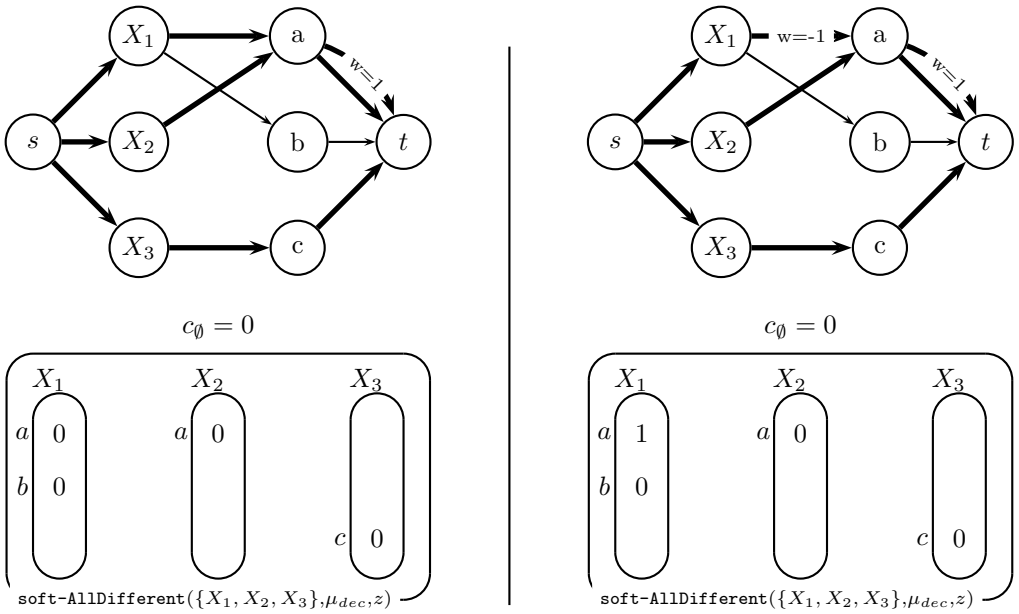
- `AllDifferent` pour les sémantiques μ_{var} et μ_{dec} ;
- `Gcc` pour les sémantiques μ_{var}, μ_{val} et μ^{Σ}_{dec}.

La contrainte `Regular` n'est pas « projection-safe », la troisième condition n'étant pas remplie.

Dans le cas de ces contraintes, il est possible de connaître la valuation minimale utilisant une valeur particulière (X_i, v_j) par la recherche d'un flot faisable de poids minimal utilisant l'arc ($X_i \rightarrow v_j$) (pour `soft-AllDifferent` voir Sous-sections 4.1.3 et 4.1.4, pour `soft-Gcc` voir Sous-sections 4.2.2 et 4.2.3, et pour Σ-`Gcc` voir Sous-section 6.1.3).

Pour maintenir en temps polynomial la cohérence globale, il est nécessaire de pouvoir transférer des coûts le long de ces contraintes globales relaxées, c'est-à-dire de pouvoir augmenter ou diminuer le coût associé à une instanciation.

Pour les contraintes, dites « projection-safe », il est possible de transférer des coûts en modifiant le poids des arcs de la partie médiane (notée $A_{\mathcal{X}}$ dans les chapitres précédents). En effet, pour transférer un coût α d'une contrainte unaire associée à l'affectation $X_i = v_j$ sur une contrainte globale, il suffit alors d'augmenter le poids de l'arc ($X_i \rightarrow v_j$) de α. La figure 7.2 présente un exemple d'un tel déplacement pour une contrainte `soft-AllDifferent` utilisant la sémantique de violation basée décomposition.



(a) Avant transfert du coût associé à $X_1 = a$ de la contrainte `soft-AllDifferent`

(b) Après transfert du coût associé à $X_1 = a$ de la contrainte `soft-AllDifferent`

FIG. 7.3 – Exemple du transfert d'un coût d'une contrainte globale vers une contrainte unaire.

De même pour une contrainte globale « projection-safe », si le coût de la meilleure instanciation complète utilisant la valeur (X_i, v_j) est α, alors il est possible de transférer ce coût α sur la contrainte unaire c_{X_i} en retirant au poids de l'arc $(X_i \rightarrow v_j)$ le poids α et en augmentant celui de $c_{X_i}(v_j)$ de α. La figure 7.3 présente un exemple d'un tel déplacement pour une contrainte `soft-AllDifferent` utilisant la sémantique de violation basée décomposition.

Des travaux similaires ont aussi été proposés pour étendre `FDAC*` en une version « généralisée » (`FDGAC*`). Pour plus de détails se référer à [LL09].

7.2 Agglomération de contraintes globales

Dans cette section, nous traitons de la seconde approche permettant de faire communiquer les contraintes entre elles en agglomérant (c'est-à-dire en englobant) plusieurs contraintes. Les contraintes globales s'inscrivent naturellement dans cette approche (la contrainte globale `All-Different` agglomère une clique de contraintes binaire de différence). Nous illustrerons cette approche au travers de la contrainte `nestedGcc` [ZP07], qui permet d'agglomérer des contraintes `Gcc` imbriquées.

7.2.1 La contrainte `nestedGcc`

i) Un exemple illustratif

Pour représenter des problèmes d'affectation de personnels avec gestion de différents niveaux de formation, on utilise généralement des contraintes `Gcc` imbriquées. Chaque contrainte `Gcc` modélisant, pour un niveau de formation, le problème d'affectation associé au personnel formé.

Pour présenter la contrainte `nestedGcc`, nous utiliserons l'exemple 25.

Exemple 25 :
Une société de téléphonie doit réaliser la maintenance de ses lignes sur trois sites a, b et c. Les réparations du site a requièrent 1 à 2 techniciens formés, de même pour le site b. Les réparations du site c ne nécessitent pas nécessairement d'employé formé mais peut en recevoir 1. Pour conduire ces réparations les sites a et c doivent recevoir 2 employés, et le site b de 1 à 4 personnes.

Parmi les 5 employés de la société, les 3 premiers ont reçu la formation nécessaire à la maintenance de ces sites (nous noterons ces employées 1, 2 et 3). Pour des raisons géographiques ces cinq employés ne peuvent interagir que sur certains sites. L'employé 1 ne peut intervenir que sur le site a, les employés 2 et 4 peuvent intervenir sur les sites a et b. L'employé 3 peut effectuer la maintenance sur les sites b et c. Et finalement, l'employé 5 ne peut intervenir que sur le site c. Le problème consiste à répartir ces 5 employés afin que les réparations soient effectuées.

À chaque employé, on associe une variable dont le domaine est l'ensemble des sites où il peut intervenir. On utilisera alors deux contraintes `Gcc` :

(G1) une première contrainte `Gcc` modélise le fait que suffisamment de personnels qualifiés se trouvent sur chaque site ;

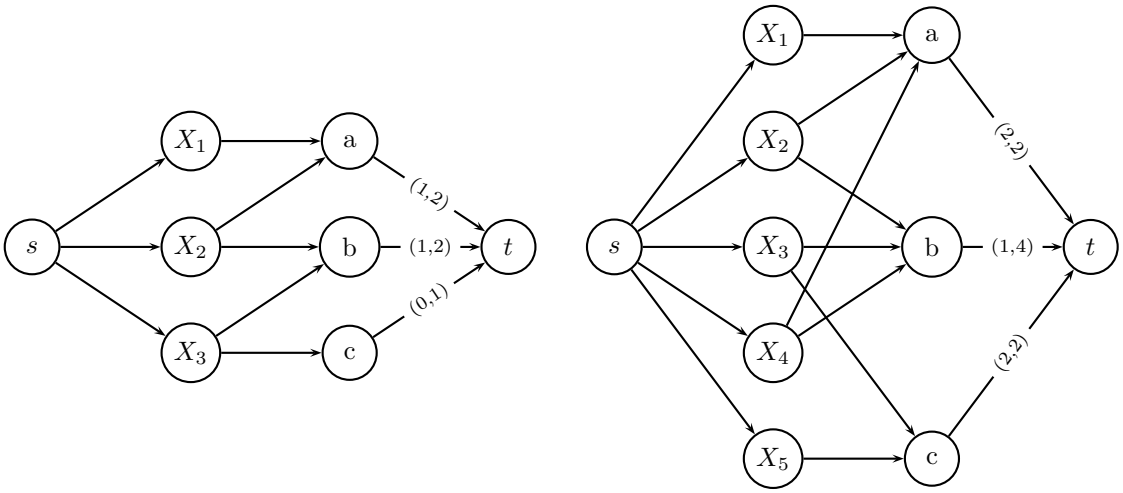
(G2) une seconde contrainte `Gcc` représente les demandes de chacun des sites en nombre d'employés.

Modélisation 3 :
Modélisation du problème de l'exemple 25 :

`//Variables et domaines`
$\mathcal{X}$=$\{X_i \mid i \in [1..5]\}$
$D_{X_1} = \{a\}$, $D_{X_2} = D_{X_4} = \{a, b\}$, $D_{X_3} = \{b, c\}$, $D_{X_5} = \{c\}$
`//Contraintes`
`Gcc`($\{X_1, X_2, X_3\}$,[1,1,0],[2,2,1]) (G1)
`Gcc`($\{X_1, X_2, X_3, X_4, X_5\}$,[2,1,2],[2,4,2]) (G2)

La figure 7.4 décrit les réseaux associés aux deux contraintes `Gcc` de l'exemple 25.

En utilisant, cette modélisation le filtrage est peu efficace. Sur notre exemple, il va falloir appeler les algorithmes de filtrage de chacune de contraintes au moins deux fois pour retirer toutes les valeurs incohérentes (dans l'ordre de retrait, les valeurs (X_3, c), (X_2, a) et (X_4, b)).



(a) Réseau de la contrainte G1 (b) Réseau de la contrainte G2

FIG. 7.4 – Les réseaux des deux contraintes `Gcc` servant à modéliser le problème de l'exemple 25.

ii) La contrainte `nestedGcc`

En tenant compte des interconnexions de ces deux contraintes `Gcc`, il aurait été possible de rendre globalement cohérent cet exemple en un seul calcul de flot grâce à la contrainte `nestedGcc` [ZP07].

Définition 104 : ***Nested global cardinality constraint***
Soit $\{\mathcal{X}^k\}_{1\leq k\leq l}$ *des ensembles disjoints de variables de* $\mathcal{X}$*,* l^k *un vecteur de bornes inférieures pour chaque* $\mathcal{X}_k$*,* u^k *un vecteur de bornes supérieures pour chaque* $\mathcal{X}_k$*, et* l *et* u *représentent respectivement les bornes inférieures et supérieures des affectations de variables de* $\mathcal{X}$*. Une contrainte* `nestedGcc`$(\mathcal{X}^1,...,\mathcal{X}^l,\mathcal{X},(l^1,u^1),...,(l^l,u^l),(l,u))$ *est cohérente* ssi *il existe une instanciation complète de* $\mathcal{X}$ *telle que :*

$$\forall k \in [1..l], \forall v_j \in Doms_{\mathcal{X}^k}, l_j^k \leq \mid \{X_i \in \mathcal{X}^k \mid X_i = v_j\} \mid \leq u_j^k$$
$$\wedge$$
$$\forall v_j \in Doms, l_j \leq \mid \{X_i \in \mathcal{X} \mid X_i = v_j\} \mid \leq u_j$$

Grâce à cette contrainte le problème de l'exemple 25 peut être modélisé en une seule contrainte `nestedGcc`. L'ensemble des techniciens formés représente un sous-ensemble de variables qui possède ses propres bornes inférieures et supérieures (voir Modélisation 4).

Modélisation 4 :
Modélisation du problème de l'exemple 25 :

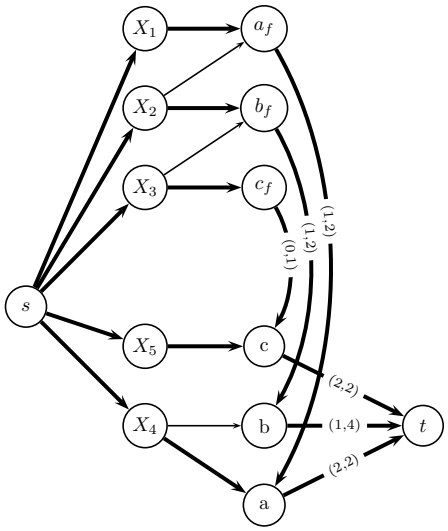


FIG. 7.5 – Réseau de la contrainte `nestedGcc` associé à l'exemple 25. Les arcs en gras décrivent un flot faisable de valeur n, donc une solution satisfaisant la contrainte `nestedGcc`.

`//Variables et domaines`
$\mathcal{X}$=$\{X_i \mid i \in [1..5]\}$
$D_{X_1} = \{a\}$, $D_{X_2} = D_{X_4} = \{a, b\}$, $D_{X_3} = \{b, c\}$, $D_{X_5} = \{c\}$
`//Contraintes`
`nestedGcc`($\{X_1, X_2, X_3\}$,$\{X_1, X_2, X_3, X_4, X_5\}$,([1,1,0],[2,2,1]),([2,1,2],[2,4,2])))

iii) Représentation sous forme d'un réseau

Il est possible de modéliser cette contrainte grâce à un réseau. Pour chaque ensemble de variables, les valeurs vont être dupliquées en des ensembles de valeurs $Doms^k$. Les variables de $\mathcal{X}^k$ et les valeurs de $Doms^k$ sont reliées comme dans le cas d'une contrainte `Gcc`. La source s est reliée à chaque variable par un arc de demande et de capacité égale à 1.

Afin de modéliser les demandes pour chaque ensemble de variables, les arcs sortant des valeurs de $Doms^k$ sont reliés aux valeurs de $Doms$ avec pour demande et capacité celles exprimées dans l^k et u^k. Finalement le dernier ensemble de variables est connecté au puits avec pour demande et capacité celles exprimées dans l et k.

La figure 7.5 représente le réseau associé à la contrainte `nestedGcc` de la seconde modélisation.

iv) Test de cohérence et filtrage

Il existe une bijection entre l'ensemble des flots faisables de valeur n dans le réseau de la contrainte `nestedGcc` et l'ensemble des instanciations complètes satisfaisant `nestedGcc`. De la même façon que pour `Gcc`, on peut :

i) tester la cohérence de la contrainte `nestedGcc` grâce à la recherche d'un flot faisable (en $O(n \times (m + n \cdot log(n)))$) grâce à l'algorithme de `Ford & Fulkerson` ;

ii) filtrer les valeurs incohérentes en calculant les composantes fortement connexes dans le réseau résiduel du flot précédemment calculé (en $O(n+m)$)) grâce à l'algorithme de `Tarjan`.

Les expériences présentées dans [ZP07] montrent que cette cohérence plus forte permet de réduire significativement le temps de calcul nécessaire pour l'obtention de solutions sur des problèmes réels (les problèmes étant issus du challenge ROADEF 2007).

Pour des cas plus complexes où les interconnexions entre les contraintes `Gcc` peuvent être décrites par un arbre, les auteurs de [ZP07] proposent une seconde contrainte `hierarchicGcc` reposant sur le même principe de connexions entre les différents réseaux des contraintes `Gcc`.

7.2.2 Autres exemples de contraintes globales agglomérantes

Les contraintes `nestedGcc` et `hierarchicGcc` ne sont pas les seules cas d'utilisation de cette approche. En effet, on peut retrouver dans la littérature les contraintes suivantes :

- la contrainte `Cardinality Matrix` [RG04] : qui permet de représenter un ensemble de contraintes de cardinalité formant une matrice $m \times n$ pour laquelle on retrouve une contrainte `Gcc` sur chaque ligne et sur chaque colonne.
- la contrainte `multi-costRegular` [MD09] : cette contrainte permet d'agglomérer plusieurs contraintes `Regular` imbriquées, ainsi que des contraintes de cardinalités (représentées sous forme d'automates).

7.3 Les contraintes `RegularCount` et `CostRegularCount`

Dans cette section, nous présentons la contrainte globale qui permet d'agglomérer une contrainte `Regular` et une contrainte de `Cardinality`[15] : `RegularCount`, ainsi que sa version relaxée : `CostRegularCount`. Ensuite nous donnons différents résultats expérimentaux comparant l'efficacité du filtrage fournit par `CostRegularCount` par rapport au filtrage séparé.

7.3.1 Motivations

Considérons l'exemple suivant :

Exemple 26 :
Soit le réseau de contraintes $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ avec l'ensemble de variables $\mathcal{X} = \{X_1, X_2, X_3, X_4, X_5, X_6\}$ ayant pour domaines $D_{X_1} = D_{X_2} = D_{X_3} = \{a\}$, $D_{X_4} = \{b\}$, $D_{X_5} = D_{X_6} = \{a, b\}$ et les deux contraintes suivante :

- `atmost`$(\mathcal{X}, a, 4)$;
- `Regular`$(\mathcal{X}, \Pi)$.

La figure 7.6 représente l'automate Π.

En utilisant de manière séparée les filtrages des contraintes `atmost` et `Regular` aucune valeur ne peut être filtrée. Or, les valeurs (X_5, a) et (X_6, a) peuvent être filtrées car elles ne peuvent

[15] La contrainte `Cardinality` est une version réduite à une valeur de la contrainte `Gcc`

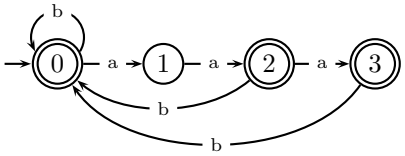


FIG. 7.6 – Automate Π.

satisfaire les contraintes `atmost` et `Regular` de manière simultanée. En effet, le nombre d'occurrence de la valeur a est limité à 4 et elle est déjà affectée trois fois. Et la contrainte `Regular` impose que la valeur a soit utilisée 2 ou 3 fois consécutivement. Cependant, en raisonnant de manière globale ces deux valeurs auraient put être filtrées.

Dans de nombreux problèmes, on retrouve l'utilisation d'une conjonction d'une contrainte `Regular`, modélisant un bon enchaînement de valeurs, et d'une contrainte `Cardinality`, contrôlant le nombre d'utilisations d'une valeur particulière. On retrouve, par exemple, les problèmes suivant :

- *Planificateur radio* : une station de radio doit insérer un certain nombre de fois un nouveau morceau dans sa planification. Afin d'éviter des transitions trop abruptes (par exemple, du classique au metal), les enchaînements musicaux doivent respecter une expression régulière. Le but est de proposer une nouvelle séquence de morceaux de musique respectant à la fois le nombre de diffusions nécessaires pour le nouveau morceau et les règles harmonieuses de transitions.
- *Affectation de personnel* : de nombreuses règles régissant le planning du personnel hospistalier peuvent être décrites grâce à un automate. De plus, les équipes de nuit sont très fatigantes et celle-ci doivent être réparties aux mieux entre le personnel d'un service.
- *Planification de période de maintenance* : le fonctionnement des machines d'un atelier répond à des suites d'opérations décrites par un automate. Afin d'assurer le bon fonctionnement d'une machine un certains nombre de périodes de maintenance doivent être intégrer au plan attribué à cette machine.

7.3.2 La contrainte `RegularCount`$(\mathcal{X}, \Pi, v, l, u)$

La contrainte `RegularCount`$(\mathcal{X}, \Pi, v, l, u)$ est définie de la façon suivante.

Définition 105 : `RegularCount`*$(\mathcal{X}, \Pi, v, l, u)$*
Soit $\Pi=\{Q, \Sigma, \delta, q_0, F\}$ *un automate fini,* $\mathcal{L}(\Pi)$ *le langage reconnu par* Π*,* $\mathcal{X}$ *un ensemble de* n *variables,* v *un symbole de l'alphabet* Σ*,* l *(resp.* u*) une borne inférieure (resp. supérieure) pour* v*. La contrainte* `RegularCount`*$(\mathcal{X}, \Pi, v, l, u)$ admet une solution* ssi *il existe une instanciation complète* $\mathcal{A}$ *telle que :*

$$\mathcal{A} \in \mathcal{L}(\Pi) \wedge l \leq |\{X_i \in \mathcal{X} \mid X_i = v\}| \leq u$$

Pour illustrer le fonctionnement du test de cohérence et de l'algorithme de filtrage de la contrainte `RegularCount`$(\mathcal{X}, \Pi, v, l, u)$, nous utiliserons l'exemple 27.

Exemple 27 :
Soit $\mathcal{X}$=$\{X_1, X_2, X_3, X_4, X_5, X_6\}$ avec pour domaines $D_{X_1} = D_{X_2} = D_{X_3} = \{a\}$, $D_{X_4} = D_{X_5} = D_{X_6} = \{a, b\}$ et la contrainte `RegularCount`$(\mathcal{X}, \Pi, a, 2, 4)$. L'automate Π est décrit dans la figure 7.6.

i) Représentation sous forme d'un graphe en couches

Dans le graphe en couches de la contrainte `Regular`$(\mathcal{X}, \Pi)$ (voir section 2.4), chaque sommet q_l^i correspond à un couple (variable X_{i+1}, état q_l) et chaque arc $(q_l^i \rightarrow q_m^{i+1})$ étiqueté par la valeur v_j représente une transition $\delta(q_l, v_j) = q_m$ dans l'automate associé.

Dans le cas de la contrainte `RegularCount`$(\mathcal{X}, \Pi, v, l, u)$, il est nécessaire de porter sur chaque sommet une nouvelle information : le nombre de fois qu'est apparue la valeur surveillée v. Ainsi dans le graphe en couches de la contrainte `RegularCount`$(\mathcal{X}, \Pi, v, l, u)$, chaque sommet $q_{l,c}^i$ représente un triplet (variable X_{i+1}, état q_l, compteur c). Les arcs de ce graphe en couches représentent toujours les transitions de l'automate associé mais doivent vérifier la propriété suivante :

Pour tout arc $a = (q_{l,c}^i \rightarrow q_{m,c'}^{i+1})$, on a $\begin{cases} c' = c + 1 & \text{si } a \text{ est étiqueté par la valeur surveillée } v \\ c' = c & \text{sinon} \end{cases}$

Les sommets de la dernière couche sont connectés au sommet t *ssi* l'état représenté par ce sommet est final et si le compteur de ce sommet respecte la contrainte de cardinalité (c'est-à-dire si $l \leq c \leq u$).

Définition 106 : ***Graphe en couches associé à*** `RegularCount`$(\mathcal{X}, \Pi, v, l, u)$
La contrainte `RegularCount`$(\mathcal{X}, \Pi, v_j, l_j, u_j)$ *peut être représentée par le graphe en couches* G=(V, A) *suivant :*

$$
\begin{array}{l}
V = \{s\} \cup V_0 \cup ... \cup V_n \cup \{t\} \\
A = A_s \cup A_1 \cup ... \cup A_n \cup A_t \\
\textit{avec} \\
\forall i \in [0..n], V_i = \{q_{l,c}^i \mid q_l \in Q, c \in [0..n]\} \\
A_s = \{(s \rightarrow q_{0,0}^0)\} \\
\forall i \in [1..n], A_i = \{(q_{l,c}^{i-1} \rightarrow q_{m,c'}^i, v_j) \mid v_j \in D_{X_i}, \delta(q_l, v_j) = q_m\} \\
\textit{avec si } v_j = v \textit{ alors } c' = c + 1 \textit{ sinon } c' = c \\
A_t = \{(q_{l,c}^n \rightarrow t) \mid q_l \in F \wedge l \leq c \leq u\}
\end{array}
$$

La figure 7.7 représente le graphe en couches associé à la contrainte `RegularCount`$(\mathcal{X}, \Pi, v, l, u)$ de l'exemple 27. Pour des raisons de clarté, seuls les sommets possédant au moins un arc sont conservés dans ce graphe en couches.

ii) Test de cohérence

Par construction du graphe en couches, la séquence d'étiquettes le long d'un s–t chemin représente un mot reconnu par Π. Seuls les sommets de la dernière couche dont le compteur est

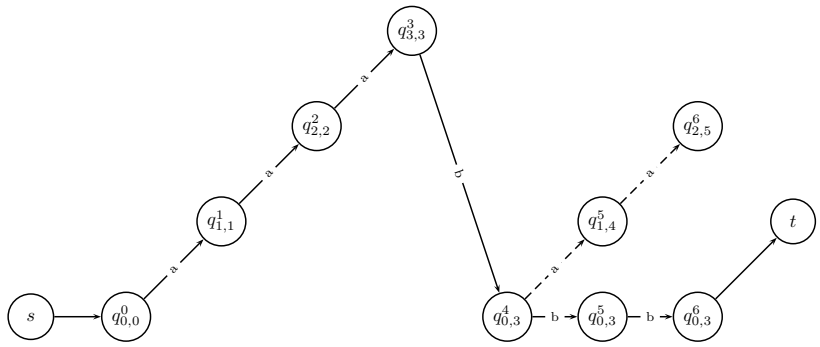


FIG. 7.7 – Graphe en couches associé à la contrainte $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$ de l'exemple 27.

compris entre l et u sont reliés à t. C'est pourquoi, tout s–t chemin correspond à une solution pour la contrainte $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$.

Théorème 12 : ***Test de cohérence de*** $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$
Une contrainte $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$ *admet une solution* ssi *il existe au moins un s–t chemin dans le graphe en couches associé.*

Le chemin $\{s, q^0_{0,0}, a, q^1_{1,1}, a, q^2_{2,2}, a, q^3_{3,3}, b, q^4_{0,3}, b, q^5_{0,3}, b, q^6_{0,3}, t\}$ dans le graphe en couches de la figure 7.7 est un s–t chemin. Celui-ci correspond à l'instanciation complète $\{(X_1 = a),(X_2 = a), (X_3 = a),(X_4 = b),(X_5 = b),(X_6 = b)\}$ qui est une solution de la contrainte $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$ de l'exemple 27.

Le test de cohérence peut être réalisé grâce à un parcours en largeur du graphe en couches en $O(n^2/2\times |Q| \times |\Sigma|)$ dans le pire cas.

iii) Filtrage

Le filtrage de la contrainte $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$ reprend le même principe que celui de la contrainte $\texttt{Regular}(\mathcal{X}, \Pi)$.

Corollaire 23 : ***Viabilité de la valeur*** (X_i, v) ***pour*** $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$
Soit la contrainte $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$*, la valeur* (X_i, v_j) *est viable* ssi *il existe un s–t chemin passant par un arc reliant les couches $i-1$ et i étiqueté par v_j.*

On peut utiliser le même algorithme de filtrage que pour la contrainte $\texttt{Regular}(\mathcal{X}, \Pi)$ et ainsi maintenir la cohérence globale en $O(n^2/2\times |Q| \times |\Sigma|)$ dans le pire cas.

Les arcs en pointillés dans la figure 7.7 représentent les arcs n'appartenant à aucun s–t chemin. Pour rendre globalement cohérente la contrainte $\texttt{RegularCount}(\mathcal{X}, \Pi, v, l, u)$ de l'exemple 27 il est nécessaire de filtrer les valeurs (X_4, a), (X_5, a) et (X_6, a).

7.3.3 La contrainte CostRegularCount($\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z$)

Afin de pouvoir traiter des problèmes sur-contraints ou dans lesquels interviennent des préférences, nous proposons la contrainte CostRegularCount($\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z$), une version relaxée avec préférences de RegularCount($\mathcal{X}, \Pi, v, l, u$).

Cette version relaxée doit tenir compte à la fois de la violation provenant de la partie « costRegular » et de la partie « Cardinality » :

- La violation provenant de la contrainte costRegular($\mathcal{X}, \Pi, z$) correspond à la somme des poids des transitions de l'automate Π utilisées pour reconnaître le mot représenté par l'instanciation des variables de $\mathcal{X}$. Nous noterons celle-ci μ_{reg}.
- La violation associée à la contrainte de Cardinality($\mathcal{X}, v, l, u$), μ_{card} va, de manière similaire à la sémantique μ^{Σ}_{dec} pour la contrainte Gcc($\mathcal{X}, l, u$), mesurer un écart aux bornes pondéré. Notons l et u les bornes associées à la valeur à surveiller, et $\varphi^{atleast}$ et φ^{atmost} les poids associés à ces bornes. La violation associée à la partie Cardinality($\mathcal{X}, v, l, u$) est définie par :

$$\mu_{card}(\mathcal{X}) = s(\mathcal{X}, v) \times \varphi^{atleast} + e(\mathcal{X}, v) \times \varphi^{atmost}$$

Ainsi, on a :

Définition 107 : ***Sémantique de violation pour la contrainte*** RegularCount***($\mathcal{X}, \Pi, v, l, u$)***
Pour une contrainte RegularCount*($\mathcal{X}, \Pi, v, l, u$), la sémantique de violation est définie par :*

$$\mu_{\overline{reg}}(\mathcal{X}) = \mu_{reg}(\mathcal{X}) + \mu_{card}(\mathcal{X})$$

Définition 108 : CostRegularCount*($\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z$)*
Soit Π=$\{Q, \Sigma, \delta, q_0, F\}$ *un automate fini pondéré,* $\mathcal{L}(\Pi)$ *le langage reconnu par* Π*,* $\mathcal{X}$ *un ensemble de* n *variables,* v *un symbole de l'alphabet* Σ*,* l *(resp.* u*) une borne inférieure (resp. supérieure) pour* v*,* $\varphi^{atleast}$ *et* φ^{atmost} *des poids associés au manque ou à l'excès de la valeur* v*, et* z *une variable de coût. La contrainte* CostRegularCount*($\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z$) admet une solution* ssi *il existe une instanciation complète* $\mathcal{A}$ *telle que :*

$$\mathcal{A} \in \mathcal{L}(\Pi) \wedge \mu_{\overline{reg}}(\mathcal{A}) \leq max(D_z)$$

Pour illustrer le fonctionnement du test de cohérence et de l'algorithme de filtrage de CostRegularCount($\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z$) nous utiliserons l'exemple 28.

Exemple 28 :
Soit $\mathcal{X}$=$\{X_1, X_2, X_3, X_4, X_5\}$ avec pour domaines $D_{X_1} = D_{X_2} = \{a\}$, $D_{X_3} = D_{X_4} = D_{X_5} = \{a, b\}$ et la contrainte CostRegularCount($\mathcal{X}, \Pi', a, 3, 3, 15, 7, z$) avec D_z=[0..10]. L'automate Π' est décrit par la figure 7.8.

i) Représentation

Le graphe en couches de la contrainte CostRegularCount($\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z$) diffère en deux points de RegularCount($\mathcal{X}, \Pi, v, l, u$) :

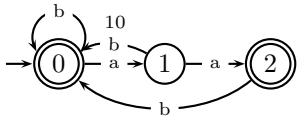


FIG. 7.8 – Automate Π'.

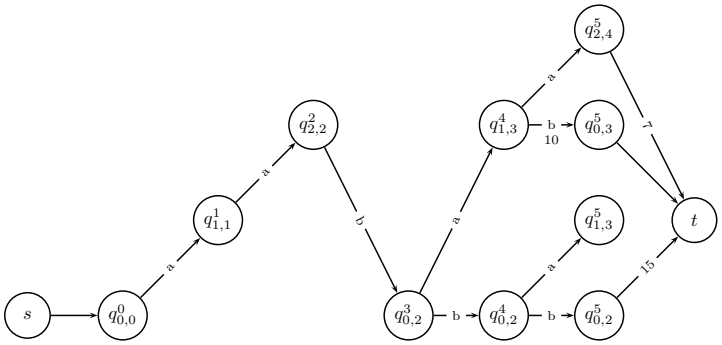


FIG. 7.9 – Graphe en couches associé à la contrainte `CostRegularCount` de l'exemple 28.

- l'automate Π est maintenant un automate avec poids ; lors de la création des différentes couches de ce graphe, les poids associés aux transitions sont maintenant reportés sur les arcs correspondants ;
- les bornes l et u peuvent, contre un coût, ne plus être respectées ; les arcs de A_t sont remplacés les arcs $A_{\bar{t}}$ suivants :

$$A_{\bar{t}} = \{(q^n_{l,c} \rightarrow t) \mid q_l \in F, c \in [0..n]\}$$
$$\textit{avec pour } a = (q^n_{l,c} \rightarrow t) \in A_{\bar{t}},$$
$$w(a) = \begin{cases} (l-c) \times \varphi^{atleast} & \textit{si } (c < l) \\ (c-u) \times \varphi^{atlmost} & \textit{si } (c > u) \\ 0 & \textit{sinon} \end{cases}$$

La figure 7.9 représente le graphe en couches associé à la contrainte `CostRegularCount`$(\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z)$ de l'exemple 28. Comme précédemment, pour des raisons de clarté, seuls les sommets possédant au moins un arc sont conservés dans ce graphe en couches.

ii) Test de cohérence

Par construction du graphe en couches, la séquence d'étiquettes le long d'un s–t chemin représente un mot reconnu par Π. De plus, le poids de ce chemin p correspond exactement au coût de violation de l'instanciation complète représentée par p.

Théorème 13 : ***Test de cohérence de*** `CostRegularCount`$(\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z)$ *Une contrainte* `CostRegularCount`$(\mathcal{X}, \Pi, v, l, u, \varphi^{atleast}, \varphi^{atmost}, z)$ *admet une solution* ssi *il existe au moins un s–t chemin p dans le graphe en couches associé tel que $weight(p) \leq max(D_z)$.*

Le chemin $\{s, q^0_{0,0}, a, q^1_{1,1}, a, q^2_{2,2}, b, q^3_{0,2}, a, q^4_{1,3}, a, q^5_{2,4}, t\}$ dans le graphe en couches de la fi-

gure 7.7 est un s–t chemin de poids 7. Celui-ci correspond à la solution $\{(X_1=a),(X_2=a),(X_3=b),(X_4=a),(X_5=a)\}$ de coût 7 pour l'exemple 28.

Le test de cohérence peut être réalisé grâce à un parcours en largeur du graphe en couches en $O(n^2/2\times|Q|\times|\Sigma|)$ dans le pire cas.

iii) Filtrage

Le filtrage s'effectue de la même façon que celui de la contrainte `RegularCount`$(\mathcal{X},\Pi,v,l,u)$.

Corollaire 24 : ***Viabilité de la valeur*** (X_i,v_j) ***pour*** `CostRegularCount`$(\mathcal{X},\Pi,v,l,u,\varphi^{atleast},\varphi^{atmost},z)$

Soit la contrainte `CostRegularCount`$(\mathcal{X},\Pi,v,l,u,\varphi^{atleast},\varphi^{atmost},z)$*, la valeur* (X_i,v_j) *est viable* ssi *il existe un* s–t *chemin* p *de poids* $weight(p)\leq max(D_z)$ *passant par un arc reliant les couches* $i-1$ *et* i *étiqueté par* v_j*.*

On peut utiliser le même algorithme de filtrage que pour la contrainte `Regular`$(\mathcal{X},\Pi)$ et maintenir la cohérence globale en $O(n^2/2\times|Q|\times|\Sigma|)$ dans le pire cas.

Afin de rendre globalement cohérente la contrainte `CostRegularCount`$(\mathcal{X},\Pi,v,l,u,\varphi^{atleast},\varphi^{atmost},z)$ de l'exemple 28 il est nécessaire de filtrer la valeur (X_4,b).

7.3.4 Expérimentations

Afin de comparer l'efficacité du filtrage de la contrainte `CostRegularCount` et du filtrage séparé, nous avons réalisé une série d'expérimentations sur un jeu d'instances de problème d'affectation de personnels. Les instances[16] sont de petite ou moyenne taille afin de pouvoir les résoudre avec une recherche arborescente complète.

Le tableau 7.1 présente, pour chaque filtrage et pour chaque instance, le temps et le nombre de backtracks nécessaires pour atteindre une solution optimale et prouver l'optimalité de celle-ci.

Instance	$\lvert\mathcal{X}\rvert$	$\lvert Doms\rvert$	UB	Σ-Gcc & `cost-regular`		`cost-regularCount`	
				temps (s.)	♯backtracks	temps (s.)	♯backtracks
inst_01_07	28	3	3000⋆	1.4	1 559	**0.2**	**342**
inst_01_11	44	3	1500⋆	20.9	14 113	**6.7**	**6 002**
inst_01_14	56	3	2500⋆	380.1	193 156	**122.6**	**63 395**
inst_02_07	49	4	1100⋆	≥5 400	–	**3 303.1**	**2 891 874**
inst_02_14	98	4	100⋆	**73.6**	24 100	120.2	**16 587**
inst_02_21	147	4	100⋆	4 886.7	1 216 908	**940.5**	**107 612**

TAB. 7.1 – Comparaison des filtrages pour la contrainte `CostRegularCount`.

Sur quasiment toutes les instances, le filtrage commun de la contrainte `CostRegularCount` diminue fortement le temps de calcul nécessaire pour trouver une solution de coût optimal et prouver son optimalité. Il permet aussi un gain très substantiel en nombre de backtracks. Sur

[16] Les instances sont décrites dans l'annexe B.

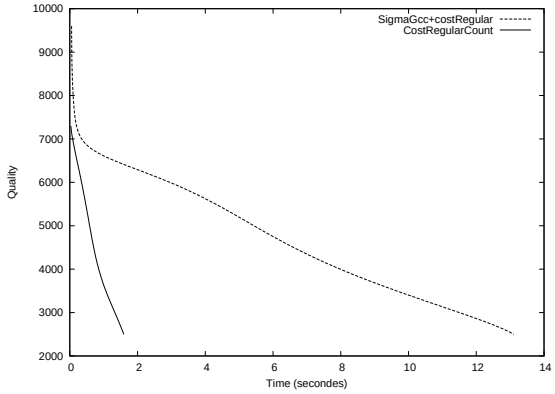


FIG. 7.10 – Comparaison de la qualité des solutions sur l'instance `inst_01_14`.

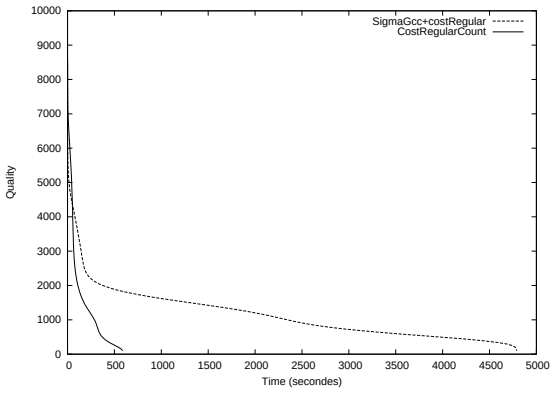


FIG. 7.11 – Comparaison de la qualité des solutions sur l'instance `inst_02_21`.

l'exemple de l'instance de plus grande taille `inst_02_21`, la contrainte `CostRegularCount` permet de trouver une solution de coût minimal (et de prouver son optimalité) cinq fois plus rapidement que le filtrage séparé.

Le filtrage séparé est plus efficace seulement dans le cas de l'instance `inst_02_14`. Cette instance est une instance facile à résoudre dans laquelle le filtrage ne joue pas un rôle crucial. Dans un tel cas, la contrainte `CostRegularCount` entraîne un léger surcoût par rapport aux contraintes Σ-`Gcc` et `costRegular`.

Les figures 7.10 et 7.11 représentent, pour les instances `inst_01_14` et `inst_02_21`, l'évolution de la qualité des instanciations complètes pour les deux types de filtrage : le filtrage commun (en trait plein) et le filtrage séparé (en pointillés).

On constate que l'utilisation de la contrainte `CostRegularCount` permet d'obtenir plus rapidement des solutions sur les deux instances considérées. Dans le cas de l'instance `inst_02_21`, durant les premières secondes du calcul, le filtrage séparé est légèrement plus rapide. Ceci est dû à la complexité plus élevée du filtrage de `CostRegularCount` qui est légèrement plus coûteux au début de la recherche.

7.4 Conclusion

La communication entre contraintes est un élément clef permettant de traiter efficacement de nombreux problèmes. Dans ce chapitre, nous avons présenté les principales approches permettant de faire communiquer des contraintes entre elles : le transport de coûts le long du réseau de contraintes et la définition de nouvelles contraintes agglomérant plusieurs contraintes.

En suivant cette deuxième approche, nous avons proposé dans ce chapitre une nouvelle contrainte permettant d'agglomérer une contrainte `Regular` et une contrainte `Cardinality`. Cette nouvelle contrainte, ainsi que sa version relaxée, exploitent la structure du graphe en couches initial et permettent, sans lourde modification, de maintenir la cohérence globale sur la conjonction de contraintes en temps polynomial (voir tableau 7.2).

Contrainte	Représentation	Cohérence	Filtrage
`RegularCount`	Graphe en couches	$O(n^2/2 \times \|Q\| \times \|\Sigma\|)$	$O(n^2/2 \times \|Q\| \times \|\Sigma\|)$
`CostRegularCount`	Graphe en couches	$O(n^2/2 \times \|Q\| \times \|\Sigma\|)$	$O(n^2/2 \times \|Q\| \times \|\Sigma\|)$

TAB. 7.2 – Nouvelles contraintes présentées dans ce chapitre.

Chapitre 8

Applications des contraintes globales relaxées aux NRPs

Sommaire

Du fait de leur complexité et de leur importance au sein des hôpitaux modernes, les « Nurse Rostering Problems » (NRPs) ont été intensivement étudiés depuis une quarantaine d'années aussi bien par la communauté Recherche Opérationnelle que par la communauté Intelligence Artificielle [BCBL04, EJKS04]. La plupart des NRPs sont des problèmes NP-complets [Kar72]. Ils sont aussi particulièrement difficiles à résoudre en raison de la grande diversité des règles et des préférences qui sont émises afin de rendre un planning acceptable en pratique. Enfin, la très grande majorité des NRPs sont des problèmes sur-contraints et difficiles à résoudre de manière efficace [aH00, QH08]

Résoudre un NRP consiste à générer un planning (ou « roster » en anglais) associant un ensemble d'équipes (Matin, Midi, Soir, ...) aux infirmières d'un service sur une période donnée, tout en respectant un ensemble de contraintes d'intégrité et de contraintes de préférence [BCBL04, BLQ09]. Ces contraintes sont généralement issues de la législation du travail, des besoins des hôpitaux ou encore des préférences émises par le personnel hospitalier. Enfin, la plupart des règles régissant le planning d'un service sont de nature globale et portent généralement sur i) l'ensemble des infirmières pour une journée donnée ou sur ii) l'ensemble des jours du planning d'une infirmière donnée.

Dans ce chapitre, nous motivons et étudions l'apport des contraintes globales relaxées pour la modélisation et la résolution des NRPs. Nous montrons que notre approche permet une modélisation élégante et concise des NRPs, et que malgré sa généricité, elle offre des temps de résolution proches de ceux obtenus par des méthodes *ad'hoc* sur de nombreuses instances issues du site du ASAP (Automated Scheduling, Optimisation and Planning) de l'université de Nottingham qui sert de référence à la communauté Recherche Opérationnelle.

8.1 Description des NRPs

Dans cette section, nous définissons le vocabulaire utile à la description d'un NRP et présentons, au travers d'un exemple représentatif, un sous-ensemble de règles issues de divers problèmes réels. Nous terminons cette section en présentant, pour cet exemple, plusieurs plannings et en décrivant pour chacun d'eux les violations induites.

8.1.1 Nurse Rostering Problems : NRPs

Résoudre un NRP consiste à construire un planning dans lequel un *ensemble d'équipes* (« shifts ») est affecté à un ensemble d'infirmières sur une période donnée appelée l'*horizon du planning* (« planning horizon ») [BCBL04, EJKS04]. Ces affectations doivent répondre aux exigences de la législation du travail, des besoins de l'hôpital et des diverses préférences émises par les infirmières d'un service ou de l'infirmière responsable de ce même service.

Un planning doit respecter :

1. des contraintes d'intégrité, par exemple, une règle imposant un nombre minimal d'infirmières pour assurer le fonctionnement d'un service ;

2. des contraintes de préférence permettant de quantifier la qualité d'un planning, par exemple, une règle exprimant une répartition équitable des temps de travail des différentes infirmières.

L'ensemble de ces contraintes (qu'elles soient d'intégrité ou de préférence) peut être divisé en deux ensembles :

- Les *contraintes d'équipes* (« shift constraints ») qui représentent des propriétés portant généralement sur le nombre d'infirmières affectables à une équipe.
- Les *contraintes d'infirmières* (« nurse constraints ») qui régissent le planning individuel d'une infirmière donnée. Ces contraintes sont généralement divisées en deux sous-ensembles :
  - les *contraintes de charges de travail* (« workload constraints ») qui limitent le nombre d'heures travaillées par une infirmière ou qui restreignent les affectations de celle-ci dans une équipe donnée ;
  - les *contraintes de séquence* (« pattern constraints ») qui définissent la « forme » des séquences de travail d'une infirmière (c'est-à-dire longueur des séquences, types d'enchaînements possibles, période de repos après une période de nuit, ...).

Un NRP comprend généralement une dizaine d'infirmières devant travailler sur une période de quatre semaines. On retrouve principalement deux ensembles d'équipe de travail :

- les « 2-shift problems », dans lesquels les infirmières travaillent dans deux équipes de douze heures : une équipe de jour D (8h00 à 20h00) et une équipe de nuit N (20h00 à 8h00) ;
- les « 3-shift problems », dans lesquels les infirmières travaillent dans des rotations de huit heures : une équipe du matin M (7h00 à 15h00), une équipe du soir E (15h00 à 23h00), et une équipe de nuit N (23h00 à 7h00).

Il existe deux types de plannings : les plannings *cycliques* et les plannings *acycliques*. Un planning cyclique est un planning qui sera répété en décalant simplement la date de début pour chacune des infirmières. Les plannings cycliques prennent difficilement en compte les exigences émises par les infirmières (comme la demande d'un jour de congé). Dans le cas des plannings acycliques, un planning est calculé pour chaque infirmière. Ainsi, les exigences des infirmières se modélisent facilement.

Remarque 19 : Dans la suite de ce chapitre, nous traiterons toutes les instances comme des plannings acycliques.

8.1.2 Un exemple de NRP

Le site du groupe de recherche du ASAP (Automated Scheduling, Optimisation and Planning) de l'université de Nottingham (`http ://www.cs.nott.ac.uk/~tec/NRP/`) recense une large collection de NRPs. L'exemple que nous présentons ici synthétise diverses règles issues de plusieurs problèmes de cette collection.

Exemple 29 :
Soit un service composé de huit infirmières pouvant travailler dans les trois équipes suivantes : l'équipe du matin M (7h00 à 15h00), l'équipe du soir E (15h00 à 23h00), et l'équipe de nuit N (23h00 à 7h00). Le but est de trouver un planning pour une période de quatre semaines qui satisfasse les contraintes d'intégrité et respecte au mieux les contraintes de préférence.

- **Contraintes d'intégrité**
  - (I1) Du lundi au vendredi, les équipes du matin, du soir et de nuit sont respectivement composées de 2, 2, et 1 infirmières.
  - (I2) Durant un week-end, les équipes du matin, du soir et de nuit sont toutes composées d'une seule infirmière.
  - (I3) Chaque infirmière doit avoir au moins dix jours de repos.
  - (I4) Chaque infirmière doit avoir au moins deux dimanches de repos.
  - (I5) Une infirmière ne doit pas travailler plus de quatre nuits.
  - (I6) Le nombre de nuits consécutives doit être compris entre deux et trois.
  - (I7) Une infirmière ne peut pas enchaîner une équipe du matin après avoir travaillé en équipe de nuit.
- **Contraintes de préférence**
  - (P1) Une infirmière doit être affectée en équipe du matin entre cinq et dix fois. Tout écart δ est pénalisé d'un coût de $\delta \times 10$.
  - (P2) Une infirmière doit être affectée en équipe du soir entre cinq et dix fois. Tout écart δ est pénalisé d'un coût de $\delta \times 10$.
  - (P3) Le nombre de jours consécutifs travaillés est au plus de quatre. Tout dépassement δ engendre une pénalité de $\delta \times 1000$.
  - (P4) Un jour de travail isolé pour une infirmière est pénalisé par un coût de 100.
  - (P5) Un jour de repos isolé pour une infirmière est pénalisé par un coût de 100.
  - (P6) Les enchaînements $E \rightarrow M$ ou $N \rightarrow E$ engendrent des pénalités de coût 100.
  - (P7) Un week-end incomplet engendre une violation de coût 200.
  - (P8) Une infirmière travaillant en équipe de nuit, un vendredi avant un week-end en repos, provoque un coût de 500.
  - (P9) Chaque infirmière doit avoir au moins trois dimanches de repos, un non respect de cette règle engendre une pénalité de 100.

Dans le cas de la règle (P3), si une infirmière est amenée à travailler six jours consécutivement alors la violation engendrée sera de 2000 ($(6-4) \times 1000$).

8.1.3 Quelques plannings associés à l'exemple

Les figures 8.1, 8.2 et 8.3 illustrent des exemples de plannings associés à l'exemple 29. Pour chacun de ces plannings, nous décrivons leurs coûts de violation.

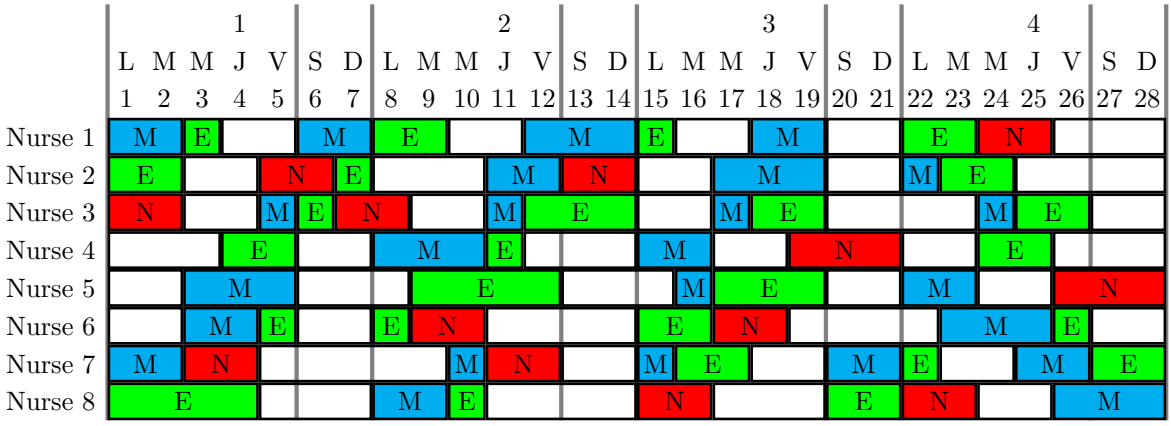


FIG. 8.1 – Un planning associé à l'exemple 29 de coût 1110.

La figure 8.1 correspond à un planning de coût 1110. Ce planning est acceptable car chaque contrainte d'intégrité est satisfaite. Le coût de 610 est dû aux violations des règles ci-dessous :

- l'infirmière 2 possède, durant le week-end du 6 au 7, un enchaînement $N \rightarrow E$ qui engendre une pénalité de 100 (P6) ;
- sur la durée de son planning, l'infirmière 3 n'effectue que quatre équipes du matin, ce qui correspond à un manque d'une équipe du matin qui est pénalisé par un coût de 10 (P1) ;
- l'infirmière 7 travaille en équipe de nuit le vendredi 12 et est en repos le week-end du 13 au 14, ce qui engendre une violation de la règle (P8) de coût 500 ;
- les infirmières 1, 2, 3, 7 et 8 travaillent toutes deux dimanches durant le planning et provoquent une violation de coût 500 (P9).

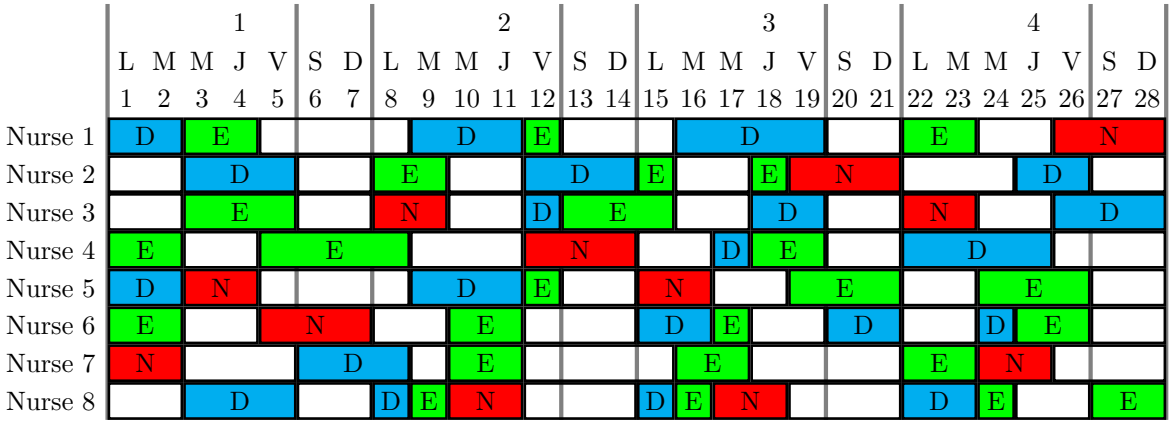


FIG. 8.2 – Un planning associé à l'exemple 29 de coût 530.

Le planning de la figure 8.2 est de meilleure qualité que celui de la figure 8.1. Son coût est de 530 car :

- sur la durée de son planning l'infirmière 2 n'effectue que quatre équipes du soir, ce qui est pénalisé par un coût de 10 par la règle (P2) ;

– sur la durée de son planning l'infirmière 7 n'effectue que trois équipes du matin, ce qui est pénalisé par un coût de 20 par la règle (P1) ;
– l'infirmière 7 possède un jour de repos isolé (jour 9), la règle (P5) engendre une violation de 100 ;
– les infirmières 2, 3, 4 et 6 travaillent toutes deux dimanches durant le planning et provoquent une violation de coût 400 (P9).

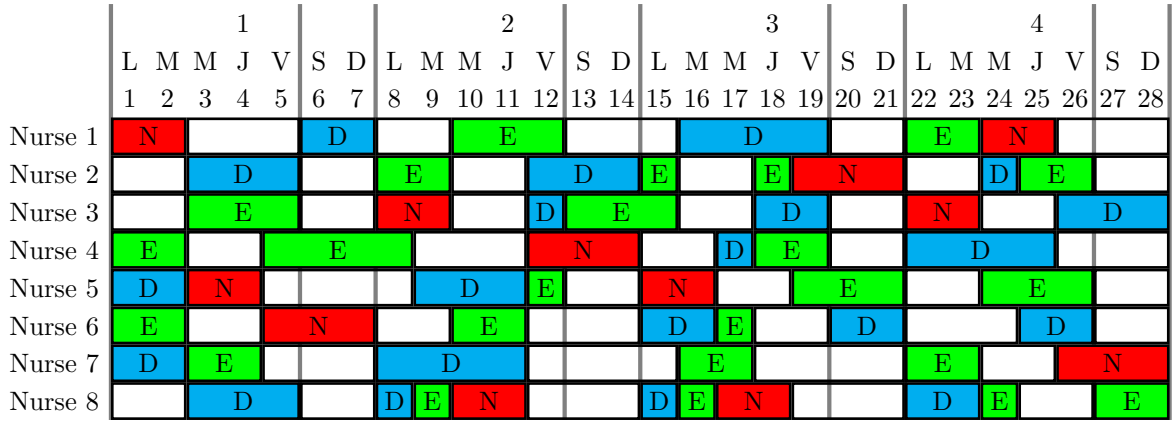


FIG. 8.3 – Un planning optimal associé à l'exemple 29.

Le dernier planning est de coût 400 :

– les infirmières 2, 3, 4 et 6 travaillent toutes deux dimanches durant le planning et provoquent une violation de coût 400 (P9).

Il s'agit d'un planning optimal.

Remarque 20 : La règle (P9) engendre nécessairement une violation de coût 400. Afin que chaque infirmière puisse avoir trois dimanches de libre, il serait nécessaire d'avoir douze infirmières dans le service, or ici le service est composé de seulement huit infirmières.

Il est à noter qu'un planning optimal n'est pas nécessairement unique. Par exemple sur le problème présenté dans la sous-section précédente, on peut remarquer que toutes les infirmières sont « interchangeables ». Ainsi, à partir de ce planning optimal, on peut calculer de nouveaux plannings optimaux grâce à des permutations.

8.2 NRPs et contraintes globales relaxées avec préférences

Dans cette section, nous motivons l'utilité des contraintes globales relaxées avec préférences pour la modélisation et la résolution des NRPs. Puis nous présentons quelles sont les contraintes qui nous semblent les mieux adaptées à la représentation de certaines règles.

8.2.1 Bref état de l'art des approches CP

Comme décrit dans [BGP03], plusieurs approches utilisant la programmation par contraintes ont été proposées pour traiter les NRPs. Parmi celles-ci, on retrouve notamment :

- des approches semi-automatiques [AS99, DFM+95] ;
- une approche utilisant une modélisation redondante pour générer les plannings hebdomadaires d'un hôpital hongkongais [CLW96] ;
- une application utilisée dans un hôpital français [CHG98] ;
- une approche basée sur une recherche locale produisant les plannings de plusieurs hôpitaux allemands [aH01] ;
- les deux applications HIBISCUS [BGP03] et GYMNASTE [Sim07] qui utilisent des contraintes globales mais ne traitent l'aspect sur-contraint que comme des perspectives.

8.2.2 Motivations

Les contraintes globales relaxées sont véritablement adaptées pour modéliser les NRPs :

1. Les règles sont souvent de *nature globale*. Elles portent généralement :
   - sur tous les jours du planning d'une infirmière ;
   - sur toutes les infirmières d'une journée.
2. Les règles sont munies de *coûts de violation exprimés de manière fine*. Ces coûts différent suivant :
   - la nature des équipes ;
   - le manque ou l'excès ;
   - les différents enchaînements d'équipes.

De plus, la puissance des algorithmes de filtrage permet une réduction efficace de l'espace de recherche.

8.2.3 Quelques contraintes globales bien adaptées

On retrouve principalement trois types de règles :

- des règles de séquence empêchant certains enchaînements d'équipes, par exemple la règle (I7) ;
- des règles restreignant les affectations aux équipes de travail comme les règles (I2) ou (P1) ;
- des règles limitant le temps de travail, qu'il soit hebdomadaire ou mensuel. Ces règles sont généralement issues de la législation du travail (la règle des 35 heures par exemple).

Chacun des types de règles peut être modélisé par une une famille de contraintes globales (relaxées ou non).

Règles de séquence

Les contraintes de la famille de `Regular` (`Regular`[Pes04], `costRegular` [DPR06], ou encore Σ-`Regular`) sont particulièrement bien adaptées à la modélisation de règles de séquence. En

effet, les enchaînements autorisés peuvent généralement être facilement modélisés sous la forme d'un automate (éventuellement pondéré) et ainsi être capturés grâce à une contrainte `Regular`. Ces règles sont des contraintes d'infirmières.

Règles restreignant les affectations aux équipes de travail

Ces règles peuvent à la fois représenter des contraintes d'équipes, comme c'est le cas de la règle (I2), ou des contraintes d'infirmières, comme par exemple la règle (I3). Elles peuvent être facilement modélisées par des contraintes de la famille de `Gcc` (`atmost`/ `atleast` [HSD92], `Gcc` [Rég96], `costGcc` [Rég99], Σ-`Gcc`). Cet ensemble de contraintes peut en effet limiter le nombre d'affectations d'une ou plusieurs équipes en fonction de bornes inférieures ou supérieures.

Règles limitant le temps de travail

Ce dernier type de règles peut être modélisé grâce à la contrainte globale `Sum` [RR00] (ou une version relaxée de celle-ci). À chaque équipe on associe une valeur numérique (représentant sa durée) et on s'assure que la somme des valeurs prises par les variables puisse être comprise entre une valeur minimale et une valeur maximale (pouvant être identiques). Ces règles sont dans la majorité des cas des contraintes d'infirmières.

8.3 Modélisation à l'aide de contraintes globales relaxées

Dans cette section, nous présentons la modélisation du problème présenté dans l'exemple 29. Soit :

- I l'ensemble des infirmières ;
- J l'ensemble des jours compris dans le planning avec :
  - J_{week} le sous-ensemble des journées du planning hors week-end ;
  - J_{we} le sous-ensemble des jours compris dans un week-end.

8.3.1 Variables et domaines

À chaque infirmière i et chaque jour j du planning, nous associons une variable X_j^i dont le domaine D_j^i contient l'ensemble des équipes dans lesquelles l'infirmière i peut travailler le jour j. Dans le cas de l'exemple 29, une infirmière peut être affectée à une équipe du matin M, du soir E ou de nuit N. Cependant, il est nécessaire d'ajouter une nouvelle équipe pour pouvoir modéliser les périodes de repos des infirmières. Nous désignerons cette équipe par la notation R. Ainsi, on a :

$$\mathcal{X} = \{X_j^i \mid i \in I, j \in J\}$$
$$\forall i \in I, \forall j \in J,\ D_{X_j^i} = \{M, E, N, R\}$$

8.3.2 Contraintes

i) Contraintes d'équipes

Les contraintes d'équipes de l'exemple 29 (règles (I1) et (I2)) sont toutes des contraintes d'intégrité et peuvent donc être modélisées par des contraintes `Gcc`. Les bornes inférieures l et supérieures u sont décrites selon l'ordre suivant : M, E, N, R.

$$\forall j \in J_{week}, \quad \texttt{Gcc}(\{X_j^1..X_j^8\}, [2,2,1,0], [2,2,1,8]) \quad (I1)$$
$$\forall j \in J_{we}, \quad \texttt{Gcc}(\{X_j^1..X_j^8\}, [1,1,1,0], [1,1,1,8]) \quad (I2)$$

Dans ces contraintes, les bornes inférieures et supérieures pour les infirmières en repos sont fixées respectivement à 0 et 8. Aucune restriction n'étant faite sur ces bornes, elles sont fixées à leur valeur extrême.

ii) Contraintes restreignant les affectations aux équipes de travail

La règle d'intégrité (I4) impose qu'une infirmière doit avoir au moins deux dimanches libres et se traduit par une contrainte `atleast`.

$$\forall i \in I, \texttt{atleast}([X_1^7, X_1^{14}, X_1^{21}, X_1^{28}], R, 2) \quad (I4)$$

Les règles (I3),(I5),(P1),(P2) peuvent être agglomérées en une seule contrainte Σ-`Gcc`(μ_{dec}^{Σ}). En effet, ces quatre règles définissent les restrictions sur le nombre de fois qu'une infirmière peut être affectée à une équipe. Seule la borne inférieure pour l'équipe de nuit, et la borne supérieure pour le nombre de repos, ne sont pas données par ces règles et sont donc fixées à leurs valeurs extrêmes.

Cette contrainte modélise à la fois des contraintes d'intégrité et des contraintes de préférence grâce à une contrainte Σ-`Gcc`(μ_{dec}^{Σ}) en donnant aux bornes des contraintes d'intégrité un poids supérieur à la valeur maximale du domaine de la variable de coût z (dans la contrainte ci-après ce poids est infini).

$$\forall i \in I, \ \Sigma\texttt{-Gcc}([X_1^i..X_{28}^i], [5,5,0,10], [10,10,4,28], [10,10,0,\infty], [10,10,\infty,0], z_i^{\alpha}) \quad (I3)(I5)(P1)(P2)$$

La règle de préférence (P9) peut se traduire par une contrainte Σ-`Gcc`(μ_{dec}^{Σ}). Comme précédemment les bornes non définies sont fixées à leurs valeurs extrêmes.

$$\forall i \in I, \ \Sigma\texttt{-Gcc}([X_1^7, X_1^{14}, X_1^{21}, X_1^{28}], [0,0,0,3], [4,4,4,4], [0,0,0,100], [0,0,0,0], z_i^{\beta}) \quad (P9)$$

Les contraintes de séquence

Le dernier ensemble de règles porte sur un ensemble d'enchaînements d'équipes qui ne doivent pas (ou le moins possible) être présents dans le planning final.

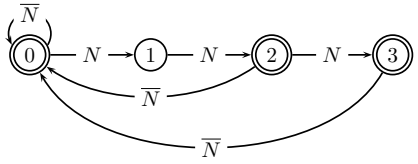


FIG. 8.4 – Automate Π_1 associé à la règle (I6).

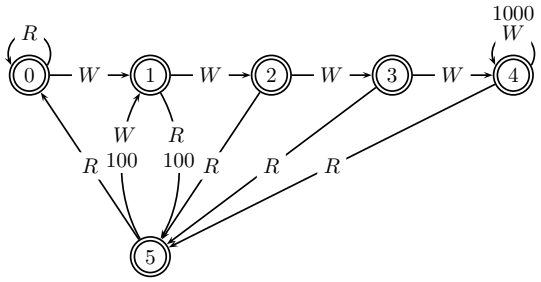


FIG. 8.5 – Automate Π_2 associé aux règles (P3), (P4) et (P5).

On retrouve notamment la contrainte d'intégrité restreignant une infirmière à devoir travailler en équipe de nuit deux ou trois fois consécutivement. La règle d'intégrité (I6) se modélise à l'aide d'une contrainte `Regular`.

$$\forall i \in I,\ \texttt{Regular}([X_1^i..X_{28}^i], \Pi_1) \quad (I6)$$

Les règles de préférence (P3), (P4) et (P5) permettent de limiter le nombre de jours consécutifs travaillés. Elles peuvent être agglomérées en un même automate et modélisées par une seule contrainte `costRegular`.

$$\forall i \in I,\ \texttt{costRegular}([X_1^i..X_{28}^i], \Pi_2, z_i^\gamma) \quad (P3), (P4), (P5)$$

L'automate de la figure 8.6 permet de modéliser à la fois une contrainte d'intégrité (I7) et une contrainte de préférence (P6) sur les enchaînements d'équipes de travail.

$$\forall i \in I,\ \texttt{costRegular}([X_1^i..X_{28}^i], \Pi_3, z_i^\delta) \quad (I7), (P6)$$

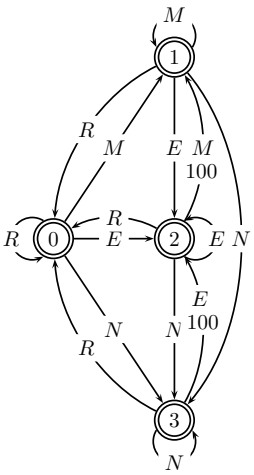


FIG. 8.6 – Automate Π_3 associé aux règles (I7) et (P6).

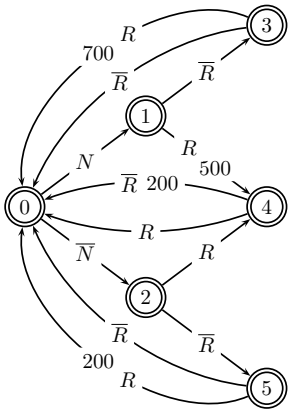


FIG. 8.7 – Automate Π_4 associé aux règles (P7) et (P8).

/ * Variables * /
$\mathcal{X} = \{X_j^i \mid i \in I, j \in J\}$
/ * Domaines * /
$\forall i \in I, \forall j \in J, D_{X_j^i} = \{M, E, N, R\}$
/ * Automates * /
automate Π_1
automate Π_2
automate Π_3
automate Π_4
/ * Contraintes * /
$\forall j \in J_{week}$,Gcc$([X_j^1..X_j^8], [2,2,1,0], [2,2,1,8])$ $(I1)$
$\forall j \in J_{we}$,Gcc$([X_j^1..X_j^8], [1,1,1,0], [1,1,1,8])$ $(I2)$
$\forall i \in I$, atleast$([X_1^7, X_1^{14}, X_1^{21}, X_1^{28}], R, 2)$ $(I4)$
$\forall i \in I$, $\Sigma-$Gcc$([X_1^i..X_{28}^i], [5,5,0,10], [10,10,4,28], [10,10,0,\infty], [10,10,\infty,0], z_i^\alpha)$ $(I3)(I5)(P1)(P2)$
$\forall i \in I$, $\Sigma-$Gcc$([X_1^7, X_1^{14}, X_1^{21}, X_1^{28}], [0,0,0,3], [4,4,4,4], [0,0,0,100], [0,0,0,0], z_i^\beta)$ $(P9)$
$\forall i \in I$, Regular$([X_1^i..X_{28}^i], \Pi_1)$ $(I6)$
$\forall i \in I$, costRegular$([X_1^i..X_{28}^i], \Pi_2, z_i^\gamma)$ $(P3), (P4), (P5)$
$\forall i \in I$, costRegular$([X_1^i..X_{28}^i], \Pi_3, z_i^\delta)$ $(I7), (P6)$
$\forall i \in I$, costRegular$([X_5^i, X_6^i, X_7^i, .., X_{26}^i, X_{27}^i, X_{28}^i], \Pi_4, z_i^\varepsilon)$ $(P7), (P8)$
/ * Variable objectif * /
Z $= \sum_{\text{i}\in\text{I}} (z_i^\alpha + z_i^\beta + z_i^\gamma + z_i^\delta + z_i^\varepsilon)$

FIG. 8.8 – Modélisation du NRP de l'exemple 29.

L'automate de la figure 8.7 est associé à la contrainte de séquence qui agglomère les deux règles associées aux week-ends.

$$\forall i \in I, \texttt{costRegular}([X_5^i, X_6^i, X_7^i, .., X_{26}^i, X_{27}^i, X_{28}^i], \Pi_4, z_i^\varepsilon) \quad (P7), (P8)$$

8.3.3 Variable objectif

La variable objectif Z correspond à la somme de toutes les variables de coûts :

$$Z = \sum_{i \in I} (z_i^\alpha + z_i^\beta + z_i^\gamma + z_i^\delta z_i^\varepsilon)$$

8.3.4 Synthèse

La modélisation d'un NRP à l'aide de contraintes globales relaxées est concise et élégante (cf. Figure 8.8). Plusieurs règles de même nature peuvent être agglomérées en une seule contrainte.

Une infirmière peut ne pas être affectée à une équipe soit parce qu'elle ne le désire pas (congé souhaité ou requête personnelle) soit parce qu'elle n'a pas la qualification nécessaire (utilisation de matériel spécifique ou responsabilité d'équipe). Dans le premier cas, il est souhaitable de représenter ce souhait comme une contrainte unaire d'égalité ou de différence. Et dans le second, il s'agit généralement d'une limitation stricte pouvant, dans la majeure partie des cas, être modélisée en retirant directement l'équipe en question du domaine des variables associées à cette infirmière.

8.4 Résolution des NRPs

De nombreuses approches ont été proposées pour résoudre les NRPs. On retrouve notamment :
- des méthodes de Recherche Opérationnelle *ad'hoc* (incluant généralement des étapes de prétraitement réduisant la taille de l'espace de recherche) ;
- des recherches locales combinées à des méthodes de Recherche Opérationnelle.

Nous avons fait le choix d'utiliser une approche générique permettant de résoudre une grande variété d'instances. Nous avons écarté les méthodes complètes utilisant la programmation par contraintes car les instances présentées dans la collection du ASAP sont soit de trop grandes tailles, soit trop difficiles pour obtenir des temps de calcul compétitifs par rapport aux méthodes *ad'hoc*. Les méthodes à grand voisinage semblent bien adaptées aux NRPs. C'est pourquoi, nous avons fait le choix d'utiliser une méthode de recherche locale de type VNS.

8.4.1 Recherche à voisinage variable (VNS)

La recherche à voisinage variable (VNS) [MH97] est une recherche à grand voisinage autorisant la taille de ces voisinages à varier afin de s'échapper de minima locaux. Pour cela, VNS utilise une structure de voisinage, c'est-à-dire un ensemble d'éléments $\{\mathcal{N}_1,...,\mathcal{N}_{kmax}\}$ ordonnés par ordre croissant de leur taille.

Variable Neighborhood Decomposition Search (VNDS) [HMPB01] étend l'idée de base de VNS. Pour toute solution de taille n, la recherche est effectuée uniquement sur un sous-problème déterminé heuristiquement. VNDS désaffecte un ensemble de k variables (parmi n) et explore le sous-espace de recherche associé à ces k variables.

VNS/LDS+CP

La méthode VNS/LDS+CP [LB08] est une extension de VNDS effectuant la reconstruction des variables grâces à une méthode arborescente partielle (LDS voir Sous-section 1.3.5) combinée avec des mécanismes de propagation (CP). L'algorithme 14 présente son pseudo-code.

L'algorithme part d'une solution initiale. Un sous-ensemble de k variables (avec k la dimension du voisinage) est alors sélectionné dans le voisinage $\mathcal{N}_k$ (c'est-à-dire l'ensemble des combinaisons de k variables parmi $\mathcal{X}$). Une affectation partielle $\mathcal{A}_p$ est alors générée à partir de la solution courante s, en désaffectant les k variables sélectionnées ; les autres variables

Algorithme 14 : Pseudo-code de l'algorithme VNS/LDS+CP.

```
1  fonction VNS/LDS+CP(X, k_init, k_max, δ)
2  début
3      s ← genPremièreSolution(X)
4      k ← k_init
5      tant que (k < k_max) ∧ (not timeout) faire
6          X_unaffected ← H_neighbor(N_k, s)
7          A_p ← s\{(X_i = a) t.q. X_i ∈ X_unaffected
8          s' ← NaryLDS(A_p, δ, cout(s), s)
9          si cout(s') < cout(s) alors
10             s ← s'
11             k ← k_init
12         sinon
13             k ← k + 1
14     retourner s
15 fin
```

conservent leur valeur dans s. Une instanciation complète est alors construite à partir de $\mathcal{A}_p$ grâce à une méthode arborescente partielle de type LDS (de discrepency δ), aidée par la propagation des contraintes (CP) afin de tirer profit de la force des mécanismes de filtrage des contraintes du problème (des contraintes globales relaxées présentées dans les chapitres précédents). Si LDS trouve une solution s' de meilleure qualité que la solution courante s, alors celle-ci devient la nouvelle solution courante et la valeur de k est réinitialisée à k_{init}. Sinon, k est incrémentée afin de s'échapper de ce minimum local. L'algorithme s'arrête dès que l'on atteint la dimension maximale du voisinage à considérer k_{max} ou un *timeout*.

8.4.2 Adaptation de VNS/LDS+CP pour les NRPs

Nous avons adapté le schéma de VNS/LDS+CP pour la résolution des NRPs de la façon suivante : si avant le *timeout* la totalité des voisinages a été visitée alors k est réinitialisée à k_{init} et la discrepency δ est augmentée de 1.

8.4.3 Solution initiale pour VNS/LDS+CP

VNS/LDS+CP a besoin d'une solution initiale afin de pouvoir commencer la recherche. Il n'est toutefois pas toujours évident de trouver une solution initiale. Dans le cas des NRPs, plusieurs alternatives sont possibles pour obtenir cette première solution. Une première possibilité est d'utiliser une solution issue de l'historique de l'hôpital (le planning du mois précédent) ou précalculée (par l'infirmière responsable du service). Une deuxième possibilité est de fournir en

entrée un planning satisfaisant seulement un sous-ensemble de contraintes (comme par exemple, les contraintes sur les capacités des équipes). Une autre possibilité est d'utiliser un solveur tiers ou une recherche arborescente, complète ou non, afin de calculer une première solution. Dans notre cas, nous utilisons LDS sur le problème initial afin de calculer un premier planning. La recherche est stoppée dès l'obtention d'une première solution.

8.4.4 Les heuristiques utilisées

Dans ces travaux, nous avons voulu essentiellement prouver l'efficacité des contraintes globales relaxées avec préférences. Par conséquent, nous avons choisi d'utiliser des heuristiques de choix de variables et de valeurs génériques et des heuristiques de choix de voisinages simples.

i) Heuristiques liées à la reconstruction

L'heuristique de choix de variables est *dom/deg* et l'heuristique de choix de valeurs est *best first* (choix de la valeur de plus petit coût de violation). Pour ces deux heuristiques, les égalités sont départagées grâce à l'ordre lexicographique.

ii) Heuristiques de choix de voisinages

Etat de l'art des heuristiques de choix de voisinages

Les heuristiques de choix de voisinages jouent un rôle crucial pour les recherches à grand voisinage. Elles guident l'exploration vers les sous-espaces de recherche les plus prometteurs. Cependant, il est très difficile de concevoir des heuristiques de choix de voisinages efficaces. Ces heuristiques sont très dépendantes du problème à traiter. De plus, comme indiqué dans [BCQB07], très peu d'heuristiques ont été proposées pour les NRPs.

Dans [BCPB04], différentes heuristiques basées sur l'échange (« swap ») de larges parties de plannings d'infirmières ont été proposées :

- *Shuffle neighborhood* qui considère des « swaps » entre les portions du planning de l'infirmière cumulant le plus de violations avec le planning d'une autre infirmière ;
- *Greedy shuffle neighborhood* qui réalise des échanges entre deux plannings d'infirmières quelconques ;
- *Core shuffle neighborhood* qui considère deux échanges consécutifs entre les plannings de deux infirmières (voir [BCPB04] pour plus de détails).

Récemment, trois nouvelles heuristiques de choix voisinages ont été proposées dans [HQ09] pour une recherche de type LNS (Large Neighborhood Search[17] [Sha98]) :

- *Fenêtre glissante de taille fixe* : les plannings de toutes les infirmières sont désinstanciés sur une fenêtre de temps donné (la taille est fixée à sept jours dans [HQ09]).

[17] LNS utilise le même mécanisme que VNS basé sur la désinstanciation et la reconstruction.

– *Fenêtre glissante avec chevauchement* : il s'agit d'une amélioration de l'heuristique précédente. Ici les variables participants à une contrainte impliquée dans la fenêtre mais qui n'ont pas déjà été selectionnées (car hors de la fenêtre) sont ajoutées aux variables à désintancier.
– *Détection des régions de base qualité* : plutôt que de sélectionner toutes les infirmières comme c'est le cas dans les deux heuristiques précédentes, la sélection porte uniquement sur celles dont le planning est de plus faible qualité.

Nos propositions

Les heuristiques de choix de voisinages basées sur l'échange ne peuvent pas être utilisées dans le cadre d'une méthode basée sur la désintanciation et la reconstruction comme `VNS/LDS+CP`.

Les heuristiques proposées pour `LNS` nous semblent peu adaptées à notre approche car :

– désinstancier les plannings de toutes les infirmières est seulement efficace pour des problèmes de petite taille. Pour des problèmes de grande taille, la taille des voisinages va rapidement grandir et requérir un effort trop important lors de leurs explorations.
– la majeure partie des contraintes globales relaxées de notre modèle ne portent pas sur une petite portion des plannings individuels mais généralement sur leur intégralité. Désinstancier une portion du planning d'une infirmière, comme dans les heuristiques « fenêtre glissante », ne permet pas une reconstruction efficace.

Toutes les variables associées à une infirmière sont fortement liées entre elles. La majorité des contraintes des `NRP`s sont des contraintes d'infirmières portant généralement sur l'intégralité du planning de celles-ci. C'est pourquoi, nous proposons de désinstancier le planning complet d'une infirmière. Nous noterons k *le nombre de plannings individuels à désinstancier* (et non pas le nombre de variables à désinstancier comme dans le schéma classique de `VNS/LDS+CP`). Les heuristiques de choix de voisinages que nous proposons sont les suivantes :

– `rand` : cette heuristique choisit aléatoirement k infirmières et désinstancie totalement leur planning ;
– `maxV` : cette heuristique désinstancie le planning des k infirmières totalisant le plus de violations ;
– `dilution` : cette heuristique combine les deux heuristiques précédentes. Parmi les k plannings à désinstancier la première moitié d'entre eux est sélectionnée par l'heuristique `maxV` et la seconde moitié par `rand`. L'idée est d'alterner des phases d'intensification (grâce aux plannings sélectionnés par `maxV`) et des phases de diversification (grâce aux plannings sélectionnés par `rand`) et ainsi de pouvoir s'échapper de minima locaux.

8.5 Expérimentations

Afin de tester l'apport des contraintes globales relaxées, nous avons réalisé des expérimentations sur une dizaine d'instances issues de la base du ASAP, `http ://www.cs.nott.ac.uk/~tex/NRP/`. Nous avons sélectionné ces instances afin d'obtenir une diversité à la fois sur la taille des instances

traitées et sur les méthodes utilisées pour les résoudre.

Pour la plupart des instances, sont répertoriées : la meilleure solution obtenue, la méthode utilisée, le temps nécessaire pour obtenir ce résultat ainsi que la machine de test.

Remarque 21 : Pour déterminer la qualité de nos résultats, nous avons comparé les solutions que nous avons obtenues avec celles des meilleures méthodes *ad'hoc* pour une instance donnée.

Les expérimentations répertoriées sur le site du ASAP ont été conduites sur un ensemble très varié de machines (allant d'un UNIVAC-1100 à un Core2Duo). Ainsi, pour chaque instance considérée, nous indiquerons le type de machine, la meilleure solution connue et le temps nécessaire pour l'obtention de celle-ci. Lorsque cela sera possible, nous indiquerons aussi un temps normalisé par rapport à notre machine de test[18]. Ces temps normalisés seront désignés par la notation CPUN.

Parmi les méthodes répertoriées sur le site du ASAP, certaines utilisent des étapes de prétraitement, dont le temps de calcul n'est pas toujours comptabilisé dans les temps reportés.

Remarque 22 : Les temps de calcul que nous reporterons pour notre méthode prennent en compte le temps nécessaire au calcul de la première solution.

8.5.1 Protocole expérimental

Les paramètres de `VNS/LDS+CP` utilisés sont les suivants :

– discrepency : $\delta_{init} = 2$;

– tailles des voisinages : $k_{init} = 2$, $k_{max} = 66\%$ du nombre total d'infirmières.

Le *timeout* a été fixé pour chaque instance en fonction de la taille de celle-ci.

Les résultats présentés pour les heuristiques de choix de voisinages `rand` et `dilution` correspondent à une moyenne sur dix exécutions.

Pour nos expérimentations, nous avons développé un solveur en C++ et réalisé les tests sur un PC équipé d'un processeur Pentium 4 cadencé à 2.8GHz et de 1Go de mémoire vive, utilisant comme système d'exploitation Linux.

8.5.2 Comparaisons avec les méthodes *ad'hoc*

Dans cette sous-section nous donnons les résultats obtenus sur les instances que nous avons sélectionnées pour nos expérimentations, et nous les comparons aux meilleurs résultats connus. Pour chaque instance, nous décrivons la ou les méthodes associées. Le tableau 8.1 synthétise ces résultats.

A) Instance `Ozkarahan` [Ozk89]

Nous trouvons l'optimum en moins d'une seconde grâce à l'heuristique `maxV`.

[18] Pour une machine κ fois plus lente que notre machine de test, le temps CPU reporté sera alors divisé par κ.

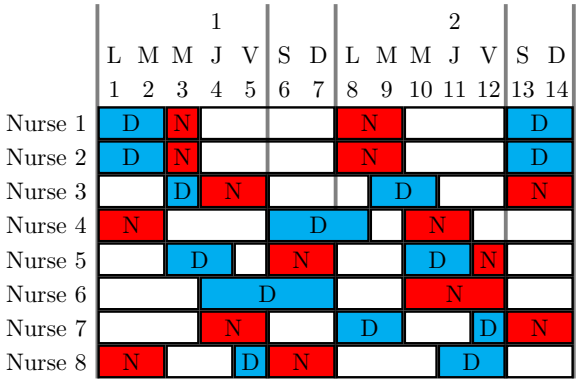


FIG. 8.9 – Solution de qualité optimale pour l'instance `Millar`.

B) Instance `Millar` [MK96]
B1 – Utilisation d'un réseau [MK96] : tous les motifs de longueur au plus quatre sont calculés. Un modèle ILP (Integer Linear Programming) est alors défini et résolu à l'aide de CPLEX.
B2 – TS+B&B (Tabu Seach + Branch and Bound) [IN03] : dans un premier temps, les contraintes d'infirmières sont utilisées pour générer tous les plannings associés aux infirmières. La meilleure combinaison de ces plannings est alors calculée en utilisant la programmation linéaire et une recherche taboue (sans tenir compte des contraintes d'équipes).

La méthode B1 trouve une solution de coût 2550 en 500 secondes sur un IBM RISC6000/340. La méthode B2 trouve une solution de coût 0 en 1 seconde sur une machine équipe d'un processeur Pentium 3 cadencé à 1GHz.

Nous trouvons une solution optimale en moins d'une seconde grâce à l'heuristique `maxV` (voir Figure 8.9).

C) Instance `Musa` [MS84]
Une solution de coût 199 est obtenue après 29 secondes de calcul, sur UNIVAC-1100.

Nous obtenons une solution optimale, de coût 175 *en* 39 *secondes, avec l'heuristique* `maxV`.

D) Instance `LLR` [LLR03]
Une approche hybridant la programmation par contraintes et une recherche taboue est utilisée dans [LLR03]. Grâce à cette méthode, une solution de coût 366 est trouvée après 96 secondes sur une machine équipée d'un processeur cadencé à 545MHz (**CPUN 16 secondes**).

Avec l'heuristique `rand`*, nous obtenons une solution de coût moyen* 316.1 *avec un timeout fixé à* 600 *secondes. La meilleure solution sur les dix exécutions est de coût* 312 *calculée en* 275 *secondes (avec* `rand`*).* Il est intéressant de noter que la qualité de la première solution est de 363 et est obtenue en moins d'une seconde.

E) L'instance `BCV-5.4.1`
Les résultats reportés dans le tableau 8.1 ont été obtenus sur une même machine équipée d'un processeur Pentium 4 2.66GHz.

La recherche tabou hybride présentée dans [BCB98] possède le meilleur temps de calcul et

Instances	$\|I\| \times \|J\|$	$\|D\|$	Optimum	Approches Ad'hoc			VNS/LDS+CP	
				Algo.	**Qualité**	**Temps (s)**	**Qualité**	**Temps (s)**
`Ozkarahan`	14×7	3	0⋆	[Ozk89]	-	-	**0**	1
`Millar`	8×14	3	0⋆	Network	2550	*500*	**0**	1
				TS+B&B	**0**	1		
`Musa`	11×14	2	175⋆	[MS84]	199	*28*	**175**	39
`LLR`	26×7	4	301⋆	TS+CP	366	16	312	275
`BCV-5.4.1`	4×28	5	48⋆	Hybrid TS	**48**	5	**48**	1
				VDS	**48**	128		
				[BBC+09]	**48**	200		
`Valouxis`	16×28	4	20⋆	VDS	60	32450	60	6570
`Azaiez`	13×28	3	0⋆	(0,1)-LGP	**0**	150	**0**	233
`GPOST_A`	8×28	3	5⋆	IP	**5**	1285	8	474
`GPOST_B`	8×28	3	3⋆	2-Phases	**3**	14	4	989
				IP	**3**	441		
`Ikegami 3Shift-DATA1`	25×30	4	2⋆	TS+B&B	6	111060	63	671

TAB. 8.1 – Tableau comparatif des approches ad'hoc vs. notre approche.

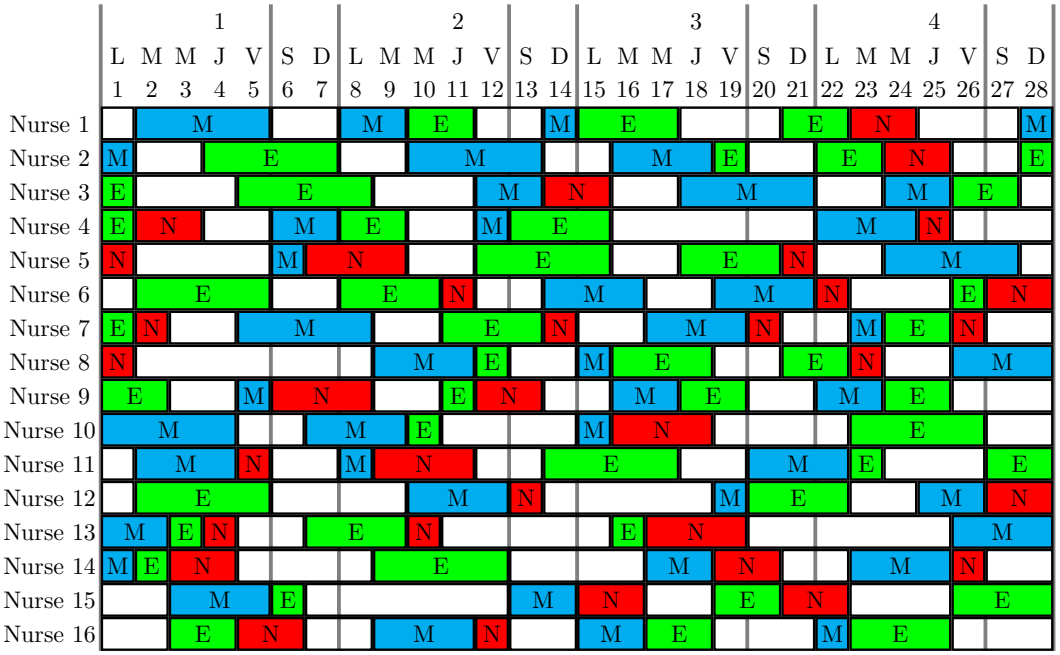


FIG. 8.10 – Solution de qualité 60 pour l'instance `Valouxis`.

obtient une solution optimale en 5 secondes (**CPUN 4 secondes**). La méthode VDS [BCQB07] trouve l'optimum après 135 secondes (**CPUN 128 secondes**). Dans [BBC+09], une solution de coût 200 est obtenue après 16 secondes (**CPUN 15 secondes**).

En utilisant `dilution`*, nous obtenons une solution de coût optimal en* 1 *seconde.*

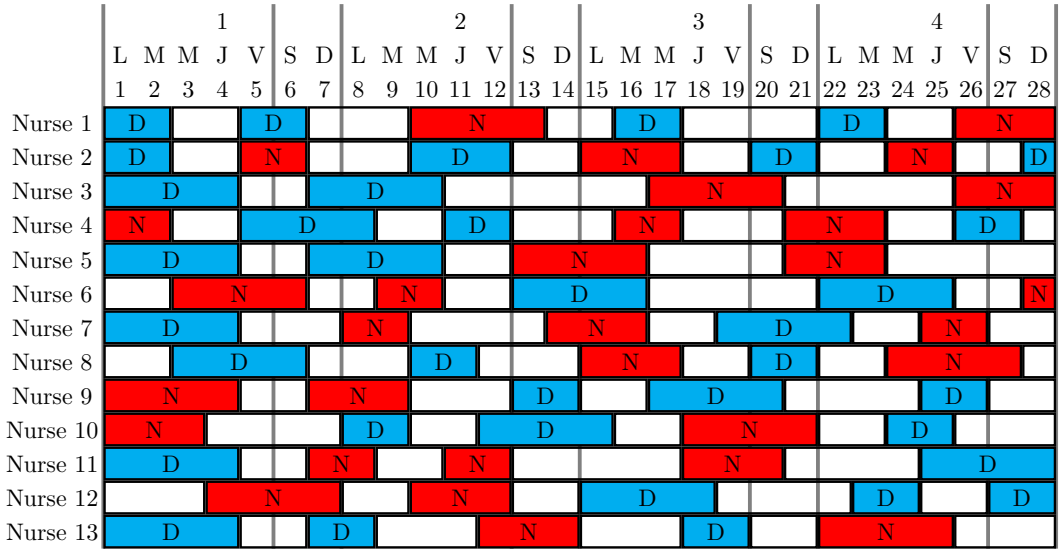


FIG. 8.11 – Solution de qualité optimale pour l'instance `Azaiez`.

F) L'instance `Valouxis` [VH00]

Dans [BCQB07], VDS obtient une solution de coût 60 après 23200 secondes de calcul sur un Core2Duo à 2.66GHz (**CPUN 32500 secondes**). VDS combine des mouvements et des échanges en utilisant différentes heuristiques pour sélectionner les prochains mouvements.

À l'aide de `rand`*, nous trouvons une solution de qualité* 60 *au bout de seulement* 6570 *secondes.*

G) L'instance `Azaiez` [AS05]

Grâce à l'utilisation de la programmation par but, une solution optimale est trouvée en 600 secondes sur un Pentium 700 MHz (**CPUN 150 secondes**).

Avec `rand`*, nous obtenons une solution optimale en un temps moyen de* 233 *secondes (pour comparaison,* `maxV` *obtient une solution en* 1050 *secondes).* La figure 8.11 illustre un des plannings optimaux obtenus.

H) Les instances `GPOST_A` et `GPOST_B`

L'instance `GPOST_A` possède une solution optimale de 5. Cet optimum a été trouvé en 1320 secondes à l'aide d'un MIP (Mixed Integer Programming) sur un Pentium4 2.66GHz (**CPUN 1285 secondes**) [GK10]. Cette approche utilise une information structurelle de l'instance qui lui permet de réduire considérablement la taille de l'espace de recherche (en déduisant un ensemble de séquences que doivent respectées les équipes de nuit).

Grâce à `dilution` *nous trouvons une solution de coût* 8 *en* 474 *secondes.*

L'instance `GPOST_B` est une version relaxée de `GPOST_A` dans laquelle les affectations des infirmières, durant les deux premiers jours, sont omises.

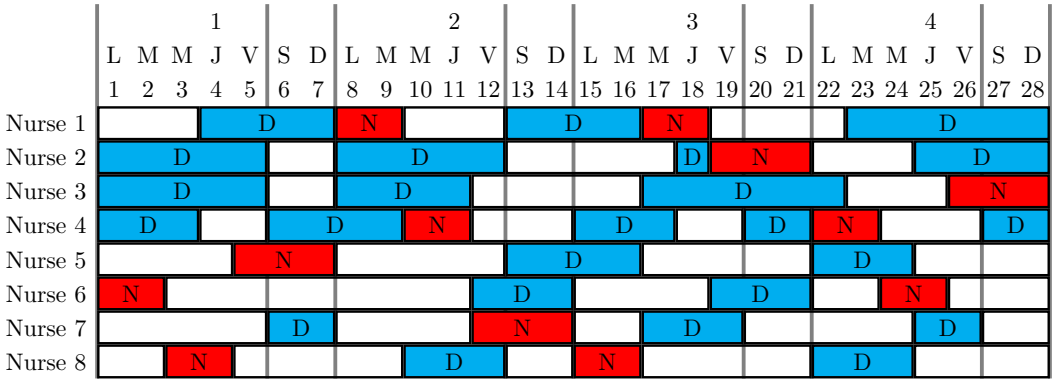


FIG. 8.12 – Solution de qualité 4 pour l'instance `GPOST_B`.

H1 – L'approche de [GK10] permet d'obtenir une solution de coût optimal de 3 en 420 secondes (**CPUN 440 secondes**).
H2 – Une approche utilisant deux phases a aussi été utilisée. Une première phase permet de générer l'ensemble des plannings individuels. Durant une seconde phase, la meilleure combinaison des plannings générés est calculée à l'aide de CPLEX. Une solution optimale de coût 3 est obtenue en 8 secondes sur un Intel Core2Duo 2.83GHz (**CPUN 14 secondes**). Ce temps de calcul ne tient pas compte du temps de calcul de la première phase.

En utilisant l'heuristique `rand`*, nous obtenons une solution de qualité* 4 *en* 989 *secondes.*

I) L'instance `Ikegami 3Shift-DATA1` [IN03]
Les expérimentations présentées dans [IN03] ont été réalisées sur un processeur Pentium 3 cadencé à 1GHz. Plusieurs solutions sont présentées en fonction du *timeout* :

- avec un *timeout* fixé à 24h, TS+B&B obtient une solution de coût 10 après 543 minutes (**CPUN 194 minutes**) ;
- avec un *timeout* fixé à 100h, TS+B&B obtient une solution de coût 6 après 5783 minutes (**CPUN 1851 minutes**).

Lors de nos premières expérimentations sur cette instance, nous avons fixé le *timeout* à 1 heure. *L'heuristique* `maxV` *obtient une solution de qualité* 63 *en* 671 *secondes sur notre machine de test.*

Remarque 23 : Une des particularités de l'instance `Ikegami` est que, contrairement aux autres instances, les contraintes d'infirmières sont des contraintes d'intégrité et les contraintes d'équipes des contraintes de préférence. Nos heuristiques de choix de voisinages qui désinstancient un planning individuel sont donc non adaptées.

Sur ce type de problème, il serait plus intéressant de désinstancier les variables qui sont associées aux contraintes d'équipes plutôt que celles associées aux contraintes d'infirmières.

Pour cette raison, nous n'avons pas poursuivi les expérimentations sur cette instance avec un *timeout* plus grand.

J) Premiers résultats sur `ORTEC_01`

L'instance `ORTEC_01` est un problème fourni par ORTEC'Harmony. Les instances `ORTEC*` sont de grande taille (31 jours pour 16 infirmières) et sont réputées comme étant parmi les plus difficiles. De très nombreuses approches ont été utilisées sur l'instance `ORTEC_01`. Le tableau 8.2 résume ces approches et les résultats obtenus par celles-ci.

Approches	Qualité	Temps (s.)
Algorithme génétique	775	3600
[BCP$^+$08]	681	86400
Heuristic ordering + VNS	706	3600
[BCP$^+$08]	681	8600
IP + VNS	541	3000
[BLQ09]		
VDS	360	1500
[BCQB07]	280	60000
MIP	270	120
[GK10]		

TAB. 8.2 – Résumé des résultats obtenus pour l'instance `ORTEC_01`.

Nous pouvons constater que l'approche utilisée dans [GK10] surclasse clairement les autres méthodes en temps et en qualité.

Lors de nos premières expérimentations, nous avons fixé le *timeout* à 7200 secondes. *Nous obtenons les résultats suivant :*

– *la meilleure solution retournée par* `rand` *est de qualité* 355 *(obtenue en* 6818 *secondes) ;*

– *la meilleure solution retournée par* `dilution` *est de qualité* 375 *(obtenue en* 4231 *secondes).*

Ces résultats préliminaires sont très prometteurs mais ont besoin d'être confirmés sur d'autres instances.

8.5.3 Comparaison des heuristiques de choix de voisinages

Nous avons également comparé expérimentalement nos trois heuristiques de choix de voisinages sur différentes instances. Le tableau 8.3 présente une synthèse des résultats. Dans ce tableau, nous reportons également la qualité de la solution initiale.

Instance	Opt.	Sol. Initial	`rand`			`maxV`		`dilution`			timeout
			Meil.	Temps	Moy.	Meil.	Temps	Meil.	Temps	Moy.	
`Millar`	0	4800	0	2	0	0	1	0	1	0	300
`BCV-5.4.1`	48	69	48	5	48.8	48	202	48	1	48.6	300
`LLR`	301	363	312	275	316.1	337	385	315	440	321.3	600
`Valouxis`	20	37240	60	6570	132	160	3780	60	7160	102	7200
`GPOST_A`	5	7876	8	654	11.4	14	1252	8	474	11	1800
`GPOST_B`	3	7362	4	989	8.5	1365	44	5	1701	8.1	1800

TAB. 8.3 – Comparaison des heuristiques `rand`, `maxV` et `dilution` sur plusieurs instances.

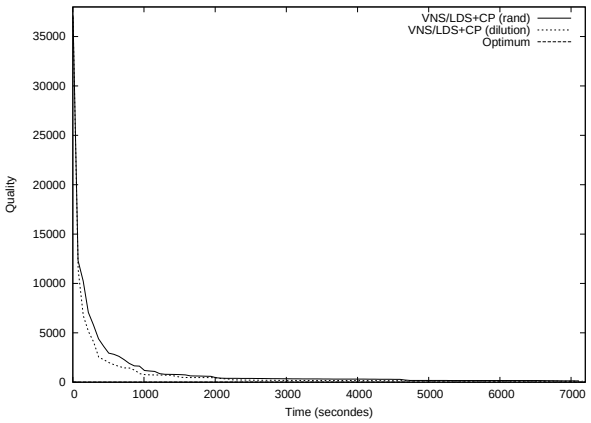


FIG. 8.13 – Évolution de la qualité en fonction du temps pour l'instance `Valouxis`.

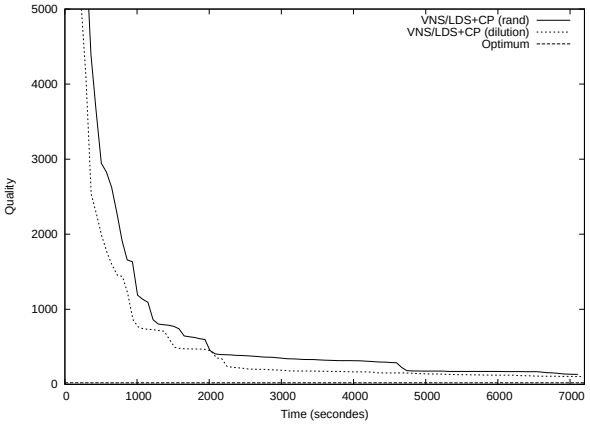


FIG. 8.14 – Zoom de l'évolution de la qualité en fonction du temps pour l'instance `Valouxis`.

Du tableau 8.3, nous tirons les remarques suivantes :

- En moyenne, l'heuristique `dilution` est meilleure que `maxV` et `rand`, sauf sur l'instance `LLR`. En effet, sur cette instance deux infirmières ont posé des repos pour la semaine complète. Or, à chaque fois qu'une infirmière a deux jours de repos consécutifs, une violation de coût 5 est engendrée (donc une violation de 30 au total). Ces deux infirmières sont donc systématiquement sélectionnées par `maxV` et ne favorisent pas l'exploration de nouveaux voisinages. L'heuristique `dilution` subit, dans une moindre mesure, les mêmes effets que `maxV`.
- Les résultats en moyenne de `dilution` et `rand` sont assez proches. En effet, lorsque la taille du voisinage devient suffissamment grande alors les voisinages deviennent très similaires.
- `maxV` est l'heuristique la moins efficace. En effet, cette heuristique est totalement déterministe, ce qui bloque rapidement la recherche dans des optima locaux. De plus, il est à noter qu'il est généralement assez difficile d'améliorer la qualité globale d'un `NRP` en ne tenant compte que des plannings les plus insatisfaits.

Nous avons également étudié l'évolution, en fonction du temps, de la qualité des solutions obtenues par les deux heuristiques : `rand` et `dilution`.

Les figures 8.13 et 8.14 montrent que, pour l'instance `Valouxis`, l'heuristique `dilution` permet d'améliorer plus rapidement la qualité des solutions durant le début de la recherche. Ce meilleur comportement peut s'expliquer par le fait que `dilution` utilise les informations de `maxV` pour améliorer la solution en guidant la recherche vers les endroits les plus insatisfaits. Le même constat peut être fait sur les courbes 8.15 et 8.16 qui présentent le comportement de ces heuristiques sur l'instance `GPOST_B`.

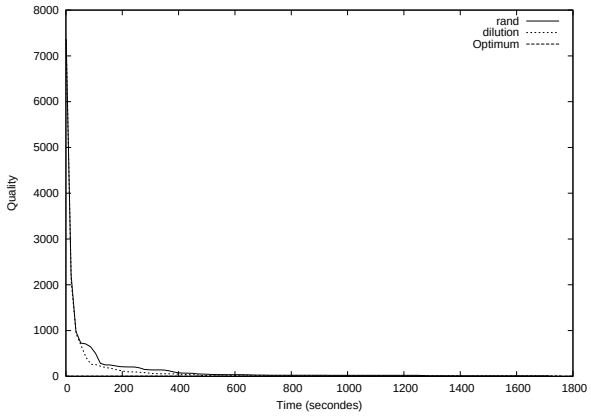


FIG. 8.15 – Évolution de la qualité en fonction du temps pour l'instance GPOST_B.

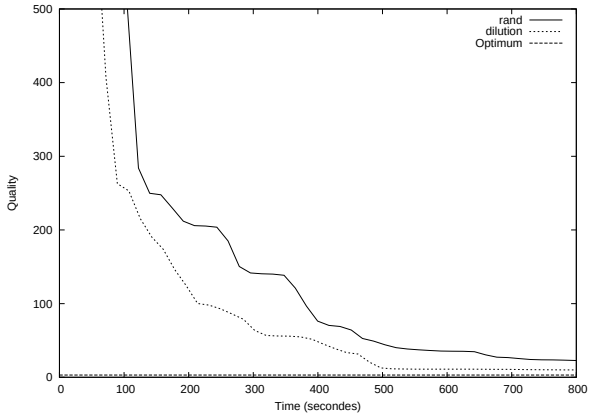


FIG. 8.16 – Zoom de l'évolution de la qualité en fonction du temps pour l'instance GPOST_B.

8.5.4 Conclusions des expérimentations

Pour chaque instance, nous avons comparé notre méthode avec la meilleure méthode *ad'hoc* utilisée pour résoudre celle-ci. Malgré la généricité et la flexibilité de notre méthode, celle-ci obtient :

- des solutions de qualité supérieure ou égale aux méthodes *ad'hoc* en des temps de calcul plus courts pour les instances Ozkarahan, Millar, Musa, LLR, BCV-5.4.1, et Valouxis ;
- des solutions de qualité égale avec des temps de calcul proches de ceux obtenus dans [AS05] pour l'instance Azaiez ;
- des résultats prometteurs sur les instances de grandes tailles comme GPOST_A, GPOST_B ou encore ORTEC_01.

8.6 Conclusion

Dans ce chapitre, nous avons montré l'apport des contraintes globales relaxées pour la modélisation et la résolution des NRPs.

Du point de vue modélisation, l'utilisation de ces contraintes globales permet d'obtenir des modèles concis (l'exemple représentatif que nous avons présenté dans la section 8.3 peut être décrit en une quinzaine de lignes).

Du point de vue résolution, l'utilisation du filtrage des contraintes globales relaxées au sein de VNS/LDS+CP a permis d'obtenir des résultats de qualité similaire à ceux obtenus par les meilleures méthodes *ad'hoc* et en des temps de résolutions comparables. Ce qui démontre la généricité et la robustesse de notre approche.

Rappelons que les expérimentations menées dans ce chapitre avait pour but de montrer l'efficacité du filtrage des contraintes globales relaxées de manière « brute », sans se focaliser sur la conception d'heuristiques sophistiquées. De nombreuses pistes d'améliorations peuvent être

envisagées pour compléter ce travail exploratoire.

Tout d'abord, il serait intéressant d'utiliser des heuristiques plus adaptées ; comme dans le cas de l'instance `Ikegami`.

De plus pour les heuristiques `rand` et `dilution`, aucune liste taboue n'est utilisée et un même voisinage (ou un voisinage très similaire) peut être exploré de manière répétée. L'utilisation d'une liste empêchant, durant une certaine durée, la désinstanciation du planning d'une infirmière, pourrait permettre d'éviter la reconstruction répétée d'un même planning individuel.

Il serait aussi intéressant d'utiliser d'autres critères que l'ordre lexicographique pour départager les égalités des heuristiques de choix de variables et de valeurs. Comme par exemple :

- une variable portant sur un jour compris dans un week-end est plus importante qu'une variable d'un jour en semaine ;
- une variable portant sur une infirmière de niveau de compétence plus élevé est prioritaire comparée à une infirmière moins qualifiée ;
- les équipes les plus contraignantes sont prioritaires.

Conclusions et perspectives

Bilan

Les travaux présentés dans ce mémoire de thèse portent sur la relaxation de contraintes globales dans un cadre permettant l'expression de préférences. Ces travaux sont, à notre connaissance, les premiers à traiter la relaxation de contraintes globales sous cet angle. En effet, les travaux antérieurs permettaient de relaxer certaines contraintes globales, en fonction de différents critères, mais sans pouvoir exprimer des préférences sur ceux-ci.

La contrainte Σ-`AllDifferent`

Dans le chapitre 5, nous avons étudié la relaxation avec préférences de la contrainte `AllDifferent`. Les deux sémantiques de violation que nous avons proposées permettent d'émettre des préférences sur les variables devant être ré-instanciées et sur les contraintes binaires de différence issues de la décomposition de la contrainte `AllDifferent`.

Dans le cas de la sémantique de violation basée variables, nous avons proposé un nouveau test de cohérence ainsi qu'un algorithme de filtrage, basé sur la recherche d'un flot, établissant la cohérence globale en un temps de calcul proche de celui de la version sans préférence.

Comme nous l'avons montré, dans le cas de la sémantique de violation basée décomposition, il n'est pas possible (sous réserve que P≠NP) de tester la cohérence globale en temps polynomial. Cependant, nous avons proposé un algorithme maintenant une cohérence plus faible en temps polynomial.

Les contraintes Σ-`Gcc` et Σ-`Regular`

Dans le chapitre 6, nous nous sommes intéressés aux contraintes `Gcc` et `Regular`. Pour ces contraintes, nous avons introduit de nouvelles sémantiques de violation autorisant l'émission de préférences. Celles-ci généralisent les travaux proposés dans [HPR06]. Pour chacune de ces sémantiques, nous avons proposé de nouveaux tests de cohérences et algorithmes de filtrages (ré-exploitant les structures prévues pour le cadre sans préférences). Ces algorithmes de filtrage établissent la cohérence globale, et cela sans surcoût important par rapport à ceux proposés pour `soft-Gcc` et `soft-Regular`.

La contrainte `RegularCount` et sa version relaxée

Nos travaux sur la résolution d'applications réelles nous ont montré que malgré la puissance des mécanismes de filtrage des contraintes globales relaxées, le manque de communication entre celles-ci peut induire une perte d'efficacité en termes de filtrage.

Dans le chapitre 7, nous avons motivé et montré l'importance d'une telle communication. Nous avons proposé la contrainte globale `RegularCount` ainsi que sa version relaxée `CostRegularCount`. Ces contraintes permettent d'agglomérer une contrainte `Regular` et une contrainte `Cardinality`. Grâce à la modification de la structure du graphe en couches de la contrainte `Regular`, nous avons proposé un test de cohérence et un algorithme de filtrage établissant la cohérence globale en temps polynomial sur la conjonction de ces deux contraintes. Les expérimentations montrent clairement que l'utilisation de `CostRegularCount` offre un gain substantiel en termes de filtrage et de temps de calcul.

Application des contraintes globales relaxées aux `NRP`s

Les contraintes globales ont été introduites pour modéliser et traiter efficacement des problèmes réels, souvent de très grande taille. Cependant, nous avons constaté que très peu d'application utilisent les contraintes globales relaxées. C'est pourquoi, nous avons tenu à proposer un cas concret d'utilisation des contraintes que nous avons introduites dans ce mémoire.

Ainsi dans le chapitre 8, nous avons présenté en quoi les contraintes globales relaxées (en particuliers Σ-`Gcc` et `costRegular`) permettent une modélisation concise et élégante et une résolution efficace des `NRP`s. L'approche que nous avons proposée est à la fois générique et robuste, elle permet de résoudre de nombreuses instances, à la différence des méthodes *ad'hoc* proposées en Recherche Opérationnelle. Tout en offrant un aspect générique, notre approche fournit des résultats très compétitifs que cela soit en termes de qualité de solutions qu'en temps de calcul. Rappelons que notre approche parvient à surpasser certaines des meilleures méthodes *ad'hoc*.

Cette application est à la fois innovante car il s'agit d'une des toutes premières utilisations de ces contraintes globales relaxées dans le cadre des problèmes d'affectation, et très prometteuse, lorsqu'on s'intéresse aux résultats produits.

Perspectives

Nos perspectives de recherche s'inscrivent dans la continuité de ces travaux de thèse : que ce soit pour améliorer les résultats fournis dans cette thèse ou pour étendre le champ d'action de ces nouvelles contraintes globales.

Un nouveau minorant pour la contrainte Σ-`AllDifferent`(μ^{Σ}_{dec})

Le minorant que nous avons proposé dans le chapitre 5 permet d'obtenir une précision assez bonne mais son calcul est relativement coûteux. Afin de permettre une utilisation efficace de la contrainte Σ-`AllDifferent`(μ^{Σ}_{dec}) dans des problèmes de grande taille (comme par exemple,

l'allocation de fréquences ou la coloration de graphes pondérés), il est nécessaire de proposer un calcul de minorant moins coûteux en temps.

L'utilisation d'une représentation sous la forme d'un réseau semble une voie très prometteuse. Le réseau (avec arcs de violation) de la contrainte `soft-AllDifferent`(μ_{dec}) pourrait être réutilisé en pondérant les arcs de violation pour modéliser les préférences émises. Afin de pondérer de manière fine les arcs de violation, certaines informations structurelles peuvent être utilisées. Par exemple, les arcs de violation issus d'une valeur ne doivent être pondérés que par les poids associés aux contraintes dont les variables possèdent cette valeur dans leur domaine. Il est aussi possible d'utiliser les variables dont le domaine est réduit à un singleton pour affiner le poids de certains arcs.

Un second avantage d'une telle représentation permettrait de s'abstraire de l'algorithme de filtrage qui est assez coûteux (rappelons que dans le pire des cas m^2 minorants seront calculés). L'utilisation d'un réseau et de la recherche d'un flot permet d'utiliser un algorithme basé sur la recherche d'un plus court chemin reliant une valeur v_j à la variable X_i afin de calculer un minorant associé à la contrainte dans le cas où l'instanciation $(X_i = v_j)$ est forcée.

Étude de sémantiques de violation non linéaires

Dans le cas des `NRP`s, on retrouve dans les instances `GPOST` et `ORTEC` la nécessité de sémantiques de violation non linéaires pour la contrainte `Gcc`. Dans ces instances, on retrouve des règles telles que la suivante :

(P) une infirmière doit travailler de quatre à cinq jours, un écart d'un jour engendre une pénalité de 1, un écart de deux jours une pénalité de 4, un écart de trois jours un coût de 9.

Une première approche pour traiter de telles sémantiques est de superposer plusieurs contraintes Σ`-Gcc` (ce qui est fait dans le chapitre 8).

Une seconde approche plus efficace consiste à « multiplier » les arcs de violation d'une contrainte Σ`-Gcc` et en donnant à ceux-ci les poids adéquats pour modéliser cette sémantique (c'est-à-dire ici un poids de 1 au premier, de 3 au deuxième et de 5 au troisième).

Cette seconde approche n'est cependant valide que dans le cas de sémantiques concaves croissantes (comme c'est le cas de n^2).

Améliorer les résultats sur les `NRP`s

Améliorer les heuristiques de reconstruction et de choix de voisinages :

Le choix que nous avons fait dans le chapitre 8 est de présenter de manière « brute » les apports des contraintes globales relaxées sur les problèmes de création d'emplois du temps d'infirmière. Pour cela, nous avons utilisé des heuristiques très génériques que cela soit pour la reconstruction ou pour l'exploration de voisinages.

Une première amélioration pourrait être d'intégrer au solveur actuel des heuristiques pour la reconstruction plus liées aux `NRP`s, telles que celles utilisées par la communauté « Automa-

ted Scheduling ». Les heuristiques proposées sont généralement très liées à la description des instances. On retrouve des heuristique de choix de variables favorisant les variables associées aux week-ends ou aux infirmières de qualification la plus élevée, et des heuristiques de choix de valeurs qui ordonnent les valeurs des variables selon un ordre prédéfini, généralement (N,E,M,R).

Comme constaté dans le chapitre 8 assez peu de travaux ont été menés pour développer des heuristiques de choix de voisinages. Le développement de nouvelles heuristiques de choix de voisinages semblent être une piste très prometteuse.

Exploiter la structure des instances :

Les travaux de [GK10] autour de l'étude structurelle des instances `GPOST` et `ORTEC` montrent qu'il est essentiel d'enrichir le modèle de nouvelles contraintes pouvant être déduites de la description des instances.

Dans un premier temps, et cela afin d'améliorer les résultats préliminaires obtenus sur ces instances, il serait intéressant d'intégrer ces nouvelles contraintes en développant une contrainte *ad'hoc* reprenant les déductions présentées dans [GK10]. Dans un second temps, il serait intéressant d'étudier les symétries présentes dans les instances. Dans la plupart des `NRP`s, un grand nombre d'infirmières peuvent être considérées comme « interchangeables ». Il est possible, assez simplement, de casser les symétries entre ces infirmières à l'aide de contraintes binaires $\leq$.

Améliorer la communication :

Le point le plus prometteur, mais aussi le plus long à mettre en place, est l'amélioration de la communication entre les contraintes globales du modèle.

Deux voies peuvent être explorées afin de diminuer cette faiblesse. La première serait l'utilisation de contraintes globales agglomérantes. Les contraintes `nestedGcc`, `Cardinality Matrix` ou encore `multi-costRegular` sont très bien adaptées à la modélisation de `NRP`s. Parmi celles-ci, la contrainte `nestedGcc` est particulièrement intéressante car on retrouve dans de nombreuses instances (parmi les plus difficiles) des `Gcc`s imbriquées afin de modéliser les demandes de certaines équipes en personnelles qualifiées. De plus, cette contrainte semble pouvoir être relaxée de la même manière que la contrainte `Gcc` grâce à l'ajout d'arcs de violation assez similaires. La seconde voie serait d'utiliser des transports de coût tels que décrit dans la section suivante.

Transport de coût dans un réseau contenant des contraintes globales relaxées

Un des constats, que l'on peut tirer des expérimentations sur les `NRP`s, est que le filtrage des contraintes globales relaxées est affaibli par le manque de communication (cf. Chapitre 7). Les contraintes globales relaxées dans le modèle de la relaxation disjonctive ne communiquent que par le biais de leur variable de coût.

Les travaux présentés dans [LL09] proposent d'étendre les mécanismes de transport de coût pour les contraintes `soft-AllDifferent`, `soft-Gcc` ou `soft-Same`. Ces travaux peuvent être immédiatement appliqués à d'autres contraintes « projection-safe » telles que `atleast`, `atmost`,

etc. Mais la proportion de contrainte globale « projection-safe » nous semble faible.

Il est essentiel d'étudier le cas des contraintes « non projection-safe ».

Vers de nouvelles applications ...

Les contraintes globales ont notamment montré leur importance grâce aux apports qu'elles fournissent sur de nombreuses applications. Pour que les contraintes globales relaxées gagnent une place similaire il est nécessaire de développer de nouvelles applications : coloration de graphes (pondérés), affectation de passagers sur des vols, planification de morceaux musicaux au sein d'une liste de diffusion, la gestion de périodes de maintenance, l'affectation de fréquences, ou des problèmes d'emplois du temps, etc.

Troisième partie

Annexes

Annexe A

Algorithme pour Hall

La détection de tous les ensembles de variables satisfaisant l'affaiblissement du théorème de Hall (voir la propriété 8 dans la section 5.3) est très couteuse. C'est pourquoi, nous nous limitons à la détection d'un sous-ensemble de ces ensembles de variables.

Ces sous-ensembles répondent à la propriété suivante :

Propriété 9 :
Les ensembles de ω variables possédant exclusivement dans leur domaine le même sous-ensemble de v valeurs, tel que $v \geq \omega$, sont des ensembles détecté par l'affaiblissement du théorème de Hall.

L'exemple suivant correspond à un tel sous-ensemble.

Exemple 30 :
Le tableau suivant présente un ensemble de variable et leur domaine.

	X_1	X_2	X_3	X_4	X_5	X_6
1	*	*	*			
2	*	*	*			
3	*	*	*			
4		*		*	*	*
5			*	*		*

Les valeurs 1, 2 et 3 forment un ensemble de trois valeurs appartenant toutes aux domaines des trois variables X_1, X_2 et X_3.

L'algorithme 15 permet de détecter de tels ensembles. Le principe de cet algorithme est de voir les variables pouvant être affectées à une valeur comme des mots binaires et de les parcourir selon l'ordre lexicographique. Si v valeurs possèdent le même ensemble de ω variables et que $v \geq \omega$ alors il s'agit d'un ensemble vérifiant l'affaiblissement du théorème de Hall.

L'étape de création des mots binaires est réalisable en $O(n \times k)$. La phase de tri nécessite un temps en $O(k \times log(k))$. Et la dernière phase requiert k comparaison de mot de longueur n,

Algorithme 15 : Algorithme détectant un sous-ensemble des ensembles de Hall.

```
1  fonction detectHall(𝒳 : un ensemble de variable) : un ensemble d'ensemble de Hall
2  début
3      ensemble ← ∅
4      // création des mots binaires
5      pour chaque v_j ∈ ⋃_{X_i∈𝒳} faire
6          pour chaque X_i ∈ 𝒳 faire
7              mot[v_j][X_i] ← X_i.getValeur(v_j)
8      // classement des mots
9      Ordonner les mot[] selon l'ordre lexicographique
10     // comparaison
11     variables ← ∅
12     nbValeur ← 0
13     precedent ← 0
14     tant que mot ≠ ∅ faire
15         (pmot, v_j) ← premier mot de mot et v_j la valeur correspondant au mot pmot
16         mot ← mot\pmot
17         si pmot == precedent ∧ mot ≠ ∅ alors
18             nbValeur ← nbValeur + 1
19         sinon
20             si |variables| ≥ nbValeur alors ensemble ← ensemble ∪ {variables}
21             variables ← {X_i | X_i ∈ 𝒳, v_j ∈ D_{X_i}}
22             nbValeur ← 1
23             precedent ← pmot
24     retourner ensemble
25 fin
```

soit $O(n \times k)$.

C'est pourquoi la détection de tels ensembles peut être fait en $O(n \times k)$.

Annexe B

Desciption des instances utilisées pour CostRegularCount

Dans cette annexe, nous décrivons les instances utilisées pour tester les performances du filtrage de la contrainte `costRegular` (voir chapitre 7).

B.1 Description des instances `inst_01_`α

Taille des instances

- Nombre d'employés : 4.
- Nombre de jours : α (avec α correspondant soit à 7, à 11, ou à 14).
- Nombre d'équipes : 3 (Jour, Nuit, Repos).

Contraintes d'équipes

- L'équipe de jour doit être composée de 2 employés.
- L'équipe de nuit doit être composée de 1 employé.

Contraintes restreignant les affectations

- Un employé doit travailler au plus 2 fois (resp. 3, 4 fois) en équipes de nuit pour un planning de 7 jours (resp. 11, 14 jours).
- Un employé doit avoir au moins 2 repos pour un planning de 7 jours (resp. 3 pour 11 jours travaillés et 4 pour 14).

Chaque écart δ engendre un coût égale à $1000 \times \delta$.

Contraintes de séquence

Violations liées à l'équipe de nuit

- Un employé ne peut être affecté en équipe de jour après avoir travaillé de nuit.

- Un employé doit travailler au moins 2 jours consécutifs en équipe de nuit (coût de 10000) et pas plus de 3 jours consécutifs (coût 10000).
- Après avoir travaillé en équipe de nuit, un employé doit avoir 2 jours de repos (coût 100).

Violations liées à la longueur des séquences de travail

- Un jour de travail isolé dans le planning est pénalisé d'un coût de 10000.
- Un employé doit travailler de 4 à 5 jours consécutivement, un écart de 1 (resp. 2) engendre une pénalité de 100 (resp. 1000).
- Un employé travaillant δ jours consécutifs (avec $\delta > 5$) engendre un coût égale à $1000 \times (\delta - 5)$.

B.2 Description des instances `inst_02_`α

Taille des instances

- Nombre d'employés : 7.
- Nombre de jours : α (avec α correspondant soit à 7, à 14, ou à 21).
- Nombre d'équipes : 4 (Matin, Soir, Nuit, Repos).

Contraintes d'équipes

Du lundi au vendredi :

- L'équipe de matin doit être composée de 2 employés.
- L'équipe du soir doit être composée de 2 employés.
- L'équipe de nuit doit être composée de 1 employé.

Durant le Week-End :

- Les équipe de matin, du soir et de nuit sont composées de 1 employé.

Contraintes restreignant les affectations

- Un employé doit travailler au plus 2 fois (resp. 4, 6 fois) en équipes de nuit pour un planning de 7 jours (resp. 14, 21 jours).
- Un employé doit avoir au moins 2 repos (resp. 4, 6 repos) et au plus 4 repos (resp. 8, 12 repos) pour un planning de 7 jours (resp. 14, 21 jours).
- Un employé doit travailler au moins 1 fois (resp. 2, 3 fois) et au plus 3 fois (resp. 6, 9 fois) pour un planning de 7 jours (resp. 14, 21 jours).

Chaque écart δ engendre un coût égale à $1000 \times \delta$.

Contraintes de séquence

Violations liées à l'équipe de nuit

– Un employé doit travailler au moins 2 jours consécutifs en équipe nuit (coût de 10000) et pas plus de 3 jours consécutifs (coût 10000).

Violations liées à la longueur des séquences de travail

– Un jour de travail isolé dans le planning est pénalisé d'un coût de 10000.
– Un employé doit travailler de 4 à 5 jours consécutivement, un écart de 1 (resp. 2) engendre une pénalité de 100 (resp. 1000).
– Un employé travaillant δ jours consécutifs (avec $\delta > 5$) engendre un coût égale à $1000 \times (\delta - 5)$.

Changement d'équipe

– Un employé ne peut être affecté en équipe de matin après avoir travaillé de nuit (coût 10000).
– Un employé ne peut être affecté en équipe de matin après avoir travaillé de soir(coût 1000).
– Un employé ne peut être affecté en équipe de soir après avoir travaillé de nuit (coût 1000).

Bibliographie

[aH00] Harald Meyer auf'm Hofe. Solving rostering tasks as constraint optimization. In Edmund K. Burke and Wilhelm Erben, editors, *PATAT*, volume 2079 of *Lecture Notes in Computer Science*, pages 191–212. Springer, 2000.

[aH01] H. Meyer auf'm Hofe. Solving rostering tasks by generic methods for constraint optimization. *Int. J. Found. Comput. Sci.*, 12(5) :671–693, 2001.

[AS99] S. Abdennadher and H. Schenker. INTERDIP - an interactive constraint based nurse scheduler. In *PACLP'99*, London, UK, 1999.

[AS05] M. Naceur Azaiez and S. S. Al Sharif. A 0-1 goal programming model for nurse scheduling. *Computers & OR*, 32 :491–507, 2005.

[BBC+09] Peter Brucker, Edmund K. Burke, Timothy Curtois, Rong Qu, and Greet Vanden Berghe. A shift sequence based approach for nurse scheduling and a new benchmark dataset. *Journal of Heuristics*, To appear :??–??, 2009.

[BC93] Christian Bessière and Marie-Odile Cordier. Arc-consistency and arc-consistency again. In *AAAI*, pages 108–113, 1993.

[BC94] Nicolas Beldiceanu and Evelyne Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12) :97–123, 1994.

[BCB98] Edmund K. Burke, Patrick De Causmaecker, and Greet Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. In Bob McKay, Xin Yao, Charles S. Newton, Jong-Hwan Kim, and Takeshi Furuhashi, editors, *SEAL*, volume 1585 of *Lecture Notes in Computer Science*, pages 187–194. Springer, 1998.

[BCBL04] Edmund K. Burke, Patrick De Causmaecker, Greet Vanden Berghe, and Hendrik Van Landeghem. The state of the art of nurse rostering. *J. Scheduling*, 7(6) :441–499, 2004.

[BCDP07] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue : Past, present and future. *Constraints*, 12(1) :21–62, 2007.

[BCP+08] Edmund K. Burke, Timothy Curtois, Gerhard F. Post, Rong Qu, and Bart Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2) :330–341, 2008.

[BCPB04] Edmund K. Burke, Patrick De Causmaecker, Sanja Petrovic, and Greet Vanden Berghe. *Variable Neighborhood Search for Rostering Problems*, pages 153–172. Kluwer Academic Publishers, 2004.

[BCQB07] Edmund K. Burke, Timothy Curtois, Rong Qu, and Greet Vanden Berghe. A time predefined variable depth search for nurse rostering. Technical Report TR 2007-6, University of Nottingham, 2007.

[BD05] Christian Bessière and Romuald Debruyne. Optimal and suboptimal singleton arc consistency algorithms. In Kaelbling and Saffiotti [KS05], pages 54–59.

[BD09] Nicolas Beldiceanu and Sophie Demassey. Global constraint catalog `http ://www.emn.fr/x-info/sdemasse/gccat/index.html`, 2009.

[Bel00] Nicolas Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. In Dechter [Dec00], pages 52–66.

[Ber70] Claude Berge. *Graphe et Hypergraphes*. Dunod, 1970.

[BFBW92] Alan Borning, Bjørn N. Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3) :223–270, 1992.

[BGP03] Stéphane Bourdais, Philippe Galinier, and Gilles Pesant. Hibiscus : A constraint programming application to staff scheduling in health care. In Francesca Rossi, editor, *CP*, volume 2833 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2003.

[BHHW07] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. The complexity of reasoning with global constraints. *Constraints*, 12(2) :239–259, 2007.

[BLQ09] Edmund K. Burke, Jingpeng Li, and Ron Qu. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research (à paraître)*, 2009.

[BMR+99] Stefano Bistarelli, Ugo Montanari, Francesca Rossi, Thomas Schiex, Gérard Verfaillie, and Hélène Fargier. Semiring-based csps and valued csps : Frameworks, properties, and comparison. *Constraints*, 4(3) :199–240, 1999.

[BP04] Nicolas Beldiceanu and Thierry Petit. Cost evaluation of soft global constraints. In Jean-Charles Régin and Michel Rueher, editors, *CPAIOR*, volume 3011 of *Lecture Notes in Computer Science*, pages 80–95. Springer, 2004.

[BR96] Christian Bessière and Jean-Charles Régin. Mac and combined heuristics : Two reasons to forsake fc (and cbj ?) on hard problems. In Eugene C. Freuder, editor, *CP*, volume 1118 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1996.

[BR01] Christian Bessière and Jean-Charles Régin. Refining the basic constraint propagation algorithm. In Bernhard Nebel, editor, *IJCAI*, pages 309–315. Morgan Kaufmann, 2001.

[CdGL+99] Bertrand Cabon, Simon de Givry, Lionel Lobjois, Thomas Schiex, and Joost P. Warners. Radio link frequency assignment. *Constraints*, 4(1) :79–89, 1999.

[CdGS07] Martin C. Cooper, Simon de Givry, and Thomas Schiex. Optimal soft arc consistency. In Manuela M. Veloso, editor, *IJCAI*, pages 68–73, 2007.

[CdGS+08] Martin C. Cooper, Simon de Givry, Martí Sánchez, Thomas Schiex, and Matthias Zytnicki. Virtual arc consistency for weighted csp. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 253–258. AAAI Press, 2008.

[CHG98] P. Chan, K. Heus, and G.Weil. Nurse scheduling with global constraints in CHIP : GYMNASTE. In *PACLP'98*, London, UK, 1998.

[CJ06] Hadrien Cambazard and Narendra Jussien. Identifying and exploiting problem structures using explanation-based constraint programming. *Constraints*, 11(4) :295–313, 2006.

[CLW96] B.M.W. Cheng, J.H.M. Lee, and J.C.K. Wu. A constraint-based nurse rostering system using a redundant modeling approach. In *Eight International Conference on Tools with Artificial Intelligence*, pages 140–148, 1996.

[Coo03] Martin C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3) :311–342, 2003.

[Cos94] Marie-Christine Costa. Persistency in maximum cardinality bipartite matchings. *Operations Research Letters*, 15(3) :143–149, 1994.

[CP97] Chia-Hsiang Chang and Robert Paige. From regular expressions to dfa's using compressed nfa's. *Theor. Comput. Sci.*, 178(1-2) :1–36, 1997.

[CS04] Martin C. Cooper and Thomas Schiex. Arc consistency for soft constraints. *Artif. Intell.*, 154(1-2) :199–227, 2004.

[Dec90] Rina Dechter. On the expressiveness of networks with hidden variables. In *AAAI*, pages 556–562, 1990.

[Dec00] Rina Dechter, editor. *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings*, volume 1894 of *Lecture Notes in Computer Science*. Springer, 2000.

[DFM+95] S. J. Darmoni, A. Fajner, N. Mahé, A. Leforestier, M. Vondracek, O. Stelian, and M. Baldenweck. Horoplan : computer-assisted nurse scheduling using constraint-based programming. *Journal of the Society for Health Systems*, 5(1) :41–54, 1995.

[DFP93] Didier Dubois, Hélène Fargier, and Henri Prade. The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. In *FUZZ-IEEE'93*, 1993.

[dG09] Simon de Givry. toulbar2 `http ://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/toolbari` 2009.

[dGHZL05] Simon de Givry, Federico Heras, Matthias Zytnicki, and Javier Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted csps. In Kaelbling and Saffiotti [KS05], pages 84–89.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik*, volume 1, pages 269–271, 1959.

[DM89] Rina Dechter and Itay Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *IJCAI*, pages 271–277, 1989.

[DP89] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artif. Intell.*, 38(3) :353–366, 1989.

[DPR06] Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4) :315–333, 2006.

[Edm65] Jack Edmonds. Path, trees, and flowers. *Canadian Journal of Mathemantics*, 17 :449–467, 1965.

[EJKS04] Andreas T. Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering : A review of applications, methods and models. *European Journal of Operational Research*, 153(1) :3–27, 2004.

[FD95] Daniel Frost and Rina Dechter. Look-ahead value ordering for constraint satisfaction problems. In *IJCAI (1)*, pages 572–578, 1995.

[FF56] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8 :399–404, 1956.

[FLS93] Hélène Fargier, Jérôme Lang, and Thomas Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In *First European Congress on Fuzzy and Intelligent Technologies*, 1993.

[FW92] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artif. Intell.*, 58(1-3) :21–70, 1992.

[GK10] C. A. Glass and R. A. Knight. The nurse rostering problem : A critical appraisal of the problem structure. *EJOR*, 202(2) :379–389, 2010.

[HDT92] Pascal Van Hentenryck, Yves Deville, and Choh-Man Teng. A generic arc-consistency algorithm and its specializations. *Artif. Intell.*, 57(2-3) :291–321, 1992.

[HE80] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3) :263–313, 1980.

[HFC+00] Martin Henz, Lim Yun Fong, Lua Seet Chong, Shi Xiao Ping, J. Paul Walser, and Roland H. C. Yap. Solving hierarchical constraints over finite domains. In *AMAI*, 2000.

[HG95] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *IJCAI (1)*, pages 607–615, 1995.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4) :225–231, 1973.

[HMPB01] Pierre Hansen, Nenad Mladenovic, and Dionisio Perez-Britos. Variable neighborhood decomposition search. *J. Heuristics*, 7(4) :335–350, 2001.

[Hoe04] Willem Jan Van Hoeve. A hyper-arc consistency algorithm for the soft alldifferent constraint. In Wallace [Wal04], pages 679–689.

[Hoe05] Willem-Jan Van Hoeve. *Operations Research Techniques in Constraint Programming*. PhD thesis, Université d'Amsterdam, April 2005.

[HPR06] Willem Jan Van Hoeve, Gilles Pesant, and Louis-Martin Rousseau. On global warming : Flow-based soft global constraints. *J. Heuristics*, 12(4-5) :347–373, 2006.

[HQ09] Fang He and Rong Qu. A constraint-directed local search approach to nurse rostering problems. *6th International Workshop on Local Search Techniques in Constraint Satisfaction*, CP'09, 2009.

[HSD92] Pascal Van Hentenryck, Helmut Simonis, and Mehmet Dincbas. Constraint satisfaction using constraint logic programming. *Artif. Intell.*, 58(1-3) :113–159, 1992.

[IN03] A. Ikegami and A. Niwa. A subproblem-centric model and approach to the nurse scheduling problem. *Mathematical Programming*, 97(3) :517–541, 2003.

[Jun04] Ulrich Junker. Quickxplain : Preferred explanations and relaxations for over-constrained problems. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 167–172. AAAI Press / The MIT Press, 2004.

[Jus03] Narendra Jussien. The versatility of using explanations within constraint programming. Habilitation thesis of Université de Nantes, 18 September 2003. also available as RR-03-04 research report at École des Mines de Nantes.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[KS05] Leslie Pack Kaelbling and Alessandro Saffiotti, editors. *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*. Professional Book Center, 2005.

[Lar02] Javier Larrosa. Node and arc consistency in weighted csp. In *AAAI/IAAI*, pages 48–53, 2002.

[Lau78] Jean-Louis Laurière. A language and a program for stating and solving combinatorial problems. *Artif. Intell.*, 10(1) :29–127, 1978.

[LB08] Samir Loudni and Patrice Boizumault. Combining VNS with constraint programming for solving anytime optimization problems. *European Journal of Operational Research*, 191(3) :705–735, 2008.

[Lev08] Nicolas Levasseur. *Heuristiques de recherche pour la résolution des WCSP*. PhD thesis, GREYC - Université de Caen Basse-Normandie, December 2008.

[LL09] Jimmy Ho Man Lee and Ka Lun Leung. Towards efficient consistency enforcement for global constraints in weighted constraint satisfaction. In *IJCAI 2009*, pages 559–565, 2009.

[LLR03] Haibing Li, Andrew Lim, and Brian Rodrigues. A hybrid ai approach for nurse rostering problem. In *SAC*, pages 730–735. ACM, 2003.

[LS03] Javier Larrosa and Thomas Schiex. In the quest of the best form of local consistency for weighted csp. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 239–244. Morgan Kaufmann, 2003.

[Mac77] Alan K. Mackworth. Consistency in networks of relations. *Artif. Intell.*, 8(1) :99–118, 1977.

[MBL07a] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. All different : Softening alldifferent in weighted csps. In *ICTAI (1)*, pages 223–230. IEEE Computer Society, 2007.

[MBL07b] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. A decomposition based softening of alldifferent in wcsps. In *CP'07 (Doctoral Program)*, pages 91–96, 2007.

[MBL08] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. Relaxation d'alldifferent avec préférences. In *JFPC'08*, pages 191–200, 2008.

[MBL09a] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. Relaxation de contraintes globales pour la modélisation de problème d'allocation d'infirmières. In *JFPC'09*, pages 155–164, 2009.

[MBL09b] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. Softening gcc and regular with preferences. In Shin and Ossowski [SO09], pages 1392–1396.

[MBL09c] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. Softening gcc with preferences. In *ROADEF'09 (Abstract)*, pages 57–58, 2009.

[MBL09d] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. Solving nurse rostering problems using soft global constraints. In Ian P. Gent, editor, *CP*, volume 5732 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2009.

[MD09] Julien Menana and Sophie Demassey. Sequencing and counting with the multicost-regular constraint. In Willem Jan Van Hoeve and John N. Hooker, editors, *CPAIOR*, volume 5547 of *Lecture Notes in Computer Science*, pages 178–192. Springer, 2009.

[MH86] Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artif. Intell.*, 28(2) :225–233, 1986.

[MH97] Nenad Mladenovic and Pierre Hansen. Variable neighborhood search. *Computers & OR*, 24(11) :1097–1100, 1997.

[MK96] H. Millar and M. Kiragu. Cyclic and non-cyclic scheduling of 12 h shift nurses by network programming. *European Journal of Operational Research*, 104(1) :582–592, 1996.

[Mon74] Ugo Montanari. Networks of constraints : Fundamental properties and applications to picture processing. *Inf. Sci.*, 7 :95–132, 1974.

[MS84] A. Musa and U. Saxena. Scheduling nurses using goal-programming techniques. *IIE transactions*, 16 :216–221, 1984.

[Ozk89] I. Ozkarahan. The zero-one goal programming model of a flexible nurse scheduling support system. In *International Industrial Engineering Conference*, pages 436–441, 1989.

[Pes04] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In Wallace [Wal04], pages 482–495.

[Pet02] Thierry Petit. *Modélisation et Algorithmes de Résolution de Problèmes Sur-Contraints*. PhD thesis, LIRMM - Université de Montpellier II, November 2002.

[PP08] Thierry Petit and Emmanuel Poder. Global propagation of practicability constraints. In Laurent Perron and Michael A. Trick, editors, *CPAIOR*, volume 5015 of *Lecture Notes in Computer Science*, pages 361–366. Springer, 2008.

[PRB00] Thierry Petit, Jean-Charles Régin, and Christian Bessière. Meta-constraints on violations for over constrained problems. In *ICTAI*, pages 358–365. IEEE Computer Society, 2000.

[PRB01] Thierry Petit, Jean-Charles Régin, and Christian Bessière. Specific filtering algorithms for over-constrained problems. In Toby Walsh, editor, *CP*, volume 2239 of *Lecture Notes in Computer Science*, pages 451–463. Springer, 2001.

[QH08] Ron Qu and Fang He. A hybrid constraint programming approach for nurse rostering problems. In *The Twenty-eighth SGAI International Conference on Artificial Intelligence*, pages 211–224, 2008.

[QLOBG04] Claude-Guy Quimper, Alejandro López-Ortiz, Peter Van Beek, and Alexander Golynski. Improved algorithms for the global cardinality constraint. In Wallace [Wal04], pages 542–556.

[QW06] Claude-Guy Quimper and Toby Walsh. Global grammar constraints. In Frédéric Benhamou, editor, *CP*, volume 4204 of *Lecture Notes in Computer Science*, pages 751–755. Springer, 2006.

[Rég94] Jean-Charles Régin. A filtering algorithm for constraints of difference in csps. In *AAAI*, pages 362–367, 1994.

[Rég95] Jean-Charles Régin. *Développement d'outils algorithmiques pour l'intelligence articielle. Application à la chimie organique*. PhD thesis, Université de Monpellier II, 1995.

[Rég96] Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. In *AAAI/IAAI, Vol. 1*, pages 209–215, 1996.

[Rég99] Jean-Charles Régin. Arc consistency for global cardinality constraints with costs. In Joxan Jaffar, editor, *CP*, volume 1713 of *Lecture Notes in Computer Science*, pages 390–404. Springer, 1999.

[Rég02] Jean-Charles Régin. Cost-based arc consistency for global cardinality constraints. *Constraints*, 7(3-4) :387–405, 2002.

[Rég04] Jean-Charles Régin. *Modélisation et Contraintes Globales en Programmation par Contraintes*. PhD thesis, Université de Nice, November 2004.

[RG04] Jean-Charles Régin and Carla P. Gomes. The cardinality matrix constraint. In Wallace [Wal04], pages 572–587.

[RR00] Jean-Charles Régin and Michel Rueher. A global constraint combining a sum constraint and difference constraints. In Dechter [Dec00], pages 384–395.

[RS59] Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2) :114–125, 1959.

[San99] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In Thomas Dean, editor, *IJCAI*, pages 542–547. Morgan Kaufmann, 1999.

[Sch92] Thomas Schiex. Possibilistic constraint satisfaction problems or "how to handle soft constraints ?". In Didier Dubois and Michael P. Wellman, editors, *UAI*, pages 268–275. Morgan Kaufmann, 1992.

[Sch00] Thomas Schiex. Arc consistency for soft constraints. In Dechter [Dec00], pages 411–424.

[Sch05] Thomas Schiex. Soft constraint processing. In *First International Summer School on Constraint Programming*, pages 1–34, 2005.

[SFV95] Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems : Hard and easy problems. In *IJCAI (1)*, pages 631–639, 1995.

[SFV97] Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Problèmes de satisfaction de contraintes valués. *Revue d'Intelligence Artificielle*, 11(3) :1–37, 1997.

[SG98] Barbara M. Smith and Stuart A. Grant. Trying harder to fail first. In *ECAI*, pages 249–253, 1998.

[Sha98] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In Michael J. Maher and Jean-Francois Puget, editors, *CP*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 1998.

[Sim07] Helmut Simonis. Models for global constraint applications. *Constraints*, 12(1) :63–92, 2007.

[SO09] Sung Y. Shin and Sascha Ossowski, editors. *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA, March 9-12, 2009*. ACM, 2009.

[Tar72] Robert Endre Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2) :146–160, 1972.

[VH00] Christos Valouxis and Efthymios Housos. Hybrid optimization techniques for the workshift and rest assignment of nursing personnel. *Artificial Intelligence in Medicine*, 20(2) :155–175, 2000.

[Wal04] Mark Wallace, editor. *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*. Springer, 2004.

[ZGS08] Matthias Zytnicki, Christine Gaspin, and Thomas Schiex. Darn ! a weighted constraint solver for rna motif localization. *Constraints*, 13(1-2) :91–109, 2008.

[ZMP06] Alessandro Zanarini, Michela Milano, and Gilles Pesant. Improved algorithm for the soft global cardinality constraint. In J. Christopher Beck and Barbara M. Smith, editors, *CPAIOR*, volume 3990 of *Lecture Notes in Computer Science*, pages 288–299. Springer, 2006.

[ZP07] Alessandro Zanarini and Gilles Pesant. Generalizations of the global cardinality constraint for hierarchical resources. In Pascal Van Hentenryck and Laurence A. Wolsey, editors, *CPAIOR*, volume 4510 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 2007.

Relaxation de contraintes globales : Mise en œuvre et Application

Dans le cadre de la Programmation par Contraintes, les contraintes globales ont amené une évolution majeure tant du point de vue modélisation (en synthétisant des ensembles de contraintes) que du point de vue résolution (grâce à des techniques de filtrage héritées d'autres domaines, comme la Recherche Opérationnelle ou l'Intelligence Artificielle). Par ailleurs, beaucoup de problèmes réels sont sur-contraints (ils ne possèdent pas de solution). Dans ce cas, il est nécessaire de relaxer certaines contraintes. De nombreuses études ont été menées pour traiter le cas des contraintes unaires et binaires, mais très peu pour le cas des contraintes globales.

Dans cette thèse, nous étudions la relaxation des contraintes globales dans un cadre permettant l'expression de préférences. Pour plusieurs contraintes globales parmi les plus utilisées (c'est-à-dire `AllDifferent`, `Gcc` et `Regular`), nous proposons différentes sémantiques de violation ainsi que des algorithmes permettant de tester l'existence d'une solution et d'éliminer les valeurs incohérentes (filtrage). Les résultats de cette thèse ont été appliqués avec succès à des problèmes de création d'emplois du temps pour des infirmières, qui sont des problèmes sur-contraints difficiles à modéliser et surtout à résoudre.

Softening Global Constraints : Implementation & Application

In Constraint Programming, global constraints have led to major changes in terms of modeling (synthesizing sets of constraints) and solving (using filtering techniques inherited from other areas such as Operational Research or Artificial Intelligence). Moreover, many real life problems are over-constrained (they have no solution). In this case, it is necessary to soften some constraints. Many studies have been conducted on unary and binary constraints, but very few on global constraints.

In this thesis, we study global constraint softening with preferences. We propose different semantics of violation for several global constraints (i.e. `AllDifferent`, `Gcc` and `Regular`). For each softening semantics, we propose algorithms to check consistency and to remove inconsistent values (filtering). The results of this thesis have been successfully applied to Nurse Rostering Problems (NRPs) that are generally over-constrained and very difficult to solve.

Mot-clefs :

- **indexation Rameau :** Programmation par contraintes, Relaxation des Contraintes, Contraintes (Intelligence Artificielle), Algorithmes, Flots (Théorie des Graphes).
- **indexation libre :** Contraintes globales, Relaxation de contraintes globales, Problèmes de création d'emplois du temps pour des infirmières.

Discipline : Informatique et applications

Groupement de Recherche en Informatique, Automatique et Instrumentation de Caen, CNRS UMR 6072
Université de Caen Basse-Normandie BP 5186
14032 Caen Cedex, FRANCE