Modélisation et contraintes globales en programmation par contraintes

Jean-Charles Régin

HABILITATION
À DIRIGER DES RECHERCHES

PRÉSENTÉE À

L'UNIVERSITÉ de NICE

ÉCOLE DOCTORALE STIC

Par **Jean-Charles Régin**

SPÉCIALITÉ : INFORMATIQUE

Modélisation et Contraintes Globales en Programmation par Contraintes

---

Soutenue le : 16 Novembre 2004

Après avis des rapporteurs :

Jacques Carlier	Professeur, Université de Technologie de Compiègne
Eugene Freuder	SFI Research Professor, University College Cork, Irlande
Pascal Van Hentenryck	Professeur, Brown University, USA

Devant la commission d'examen composée de :

Yves Caseau	Directeur Central des Systèmes d'Information, Bouygues Telecom
Jacques Carlier	Professeur, Université de Technologie de Compiègne
Alain Colmerauer	Professeur, Université de la Méditerranée
Michel Cosnard	Directeur de l'Unité de Recherche INRIA Sophia Antipolis
François Fages	Directeur de Recherche, INRIA Rocquencourt
Michel Rueher	Professeur, Université de Nice

Table des matières

Avant propos

Le but de ce document n'est pas de faire une synthèse des travaux récents en PPC, mais de mettre en évidence mes contributions dans ce domaine. Je ne parlerai donc pas de nombreux aspects de la PPC comme la détection et l'élimination des symétries ou les méthodes de recherche car je n'ai publié aucun article sur ces sujets. Je ne parlerai pas non plus de certains domaines importants de la PPC comme les CSP numériques ou l'ordonnancement pour lesquels un travail considérable a été effectué.

Ce document présente les travaux les plus importants que j'ai publiés et quelques résultats originaux non encore publiés. Il est structuré en six parties :

- Présentation de la PPC.
- Principes de la PPC et problématique de la modélisation.
- Algorithmes de filtrage et contraintes globales.
- Problèmes sur-contraints.
- Applications.
- Réflexions et perspectives.

Brièvement, on peut donner une idée du contenu de chaque partie :

Présentation de la PPC : Cette partie commence par une présentation didactique de la PPC, puis montre différents domaines d'application de cette technique. Enfin, les liens entre PPC et Recherche Opérationnelle sont étudiés.

Principes de la PPC et problématique de la modélisation : Cette partie propose une étude des principes de la PPC de façon détaillée en mettant l'accent sur les problèmes de modélisation que l'on rencontre en PPC. La notion de contrainte globale, fondamentale en PPC, est introduite et un exemple de modélisation efficace est donné.

Algorithmes de filtrage et contraintes globales : Cette partie commence par présenter des algorithmes génériques de filtrage dans le cas de contraintes binaires et pour les contraintes non binaires. Puis, il est montré comment certains algorithmes de filtrage efficaces sont obtenus en intégrant des algorithmes de Recherche Opérationnelle et de théorie des graphes. Les contraintes alldiff, symmetric alldiff, de cardinalité globale sans et avec coûts sont étudiées. Ensuite, certaines contraines dérivées sont proposées, notamment la contraine alldiff avec coûts, la contrainte de plus courts chemins, la contrainte k-diff et les contraintes ensemblistes partition et allNullIntersect. Cette partie se termine par l'étude de deux autres contraintes globales : la contrainte combinant une somme et des inégalités binaires et la contrainte globale de séquence.

Problèmes sur-contraints : Après une étude des méthodes existantes et de la mise en évidence de certains de leurs inconvénients pour résoudre des problèmes réels, cette partie propose une nouvelle méthode de modélisation des problèmes sur-contraints. Le problème particulier de la minimisation du nombre de contraintes violées est alors considéré et plusieurs algorithmes de filtrage sont proposés dont l'un est original et n'a jamais été publié. Pour finir, cette partie présente deux définitions générales du coût de violation d'une contrainte globale et introduit la notion de contrainte globale molle au travers de la contrainte alldiff molle.

Applications : Une étude approfondie de la résolution en PPC de deux applications est considéré dans cette partie : la recherche de la taille de la plus grande clique dans un graphe et le problème du dimensionnement d'un réseau de télécommunication.

Réflexions et perspectives : Bien que ce document propose régulièrement des perspectives de recherche, cette partie propose de s'attarder sur deux thèmes : la qualité d'un algorithme de filtrage et la problématique de l'approche constructive qui s'oppose aux méthodes habituelles basées sur la notion de filtrage.

Chapitre 1

Présentation de la PPC

La programmation par contraintes (PPC) est une technique de résolution de problèmes qui vient de l'intelligence artificielle et qui est née dans les années 70 du siècle dernier.

On peut dire que la PPC telle qu'elle existe s'est principalement inspirée :

- des travaux sur les "General Problem Solvers", notamment ceux de J-L. Laurière qui a proposé le sytème ALICE [Laurière, 1976, Laurière, 1978]
- de la programmation logique avec contraintes et principalement de Prolog [Colmerauer, 1990] et des travaux de P. van Hentenryck qui a implémenté Alice en Prolog [Van Hentenryck, 1989] pour donner naissance au langage CHIP (Constraint Handling In Prolog).
- des problèmes de satisfaction de contraintes (CSP) [Waltz, 1975, Montanari, 1974, Mackworth, 1977, Freuder, 1978].

C'est avec le premier de ces trois domaines que la PPC est le plus proche. En effet, la programmation logique avec contrainte est basée sur Prolog et la théorie des CSP reste peu intéressée par la représentation du problème.

En Programmation Par Contraintes (PPC), un problème est défini à partir de variables et de contraintes. Chaque variable est munie d'un domaine définissant l'ensemble des valeurs possibles pour cette variable. Une contrainte exprime une propriété qui doit être satisfaite par un ensemble de variables.

En PPC, un problème est aussi vu comme une conjonction de sous-problèmes pour lesquels on dispose de méthodes efficaces de résolution. Ces sous-problèmes peuvent être très simples comme $x < y$ ou complexe comme la recherche d'un flot compatible. Ces sous-problèmes correspondent aux contraintes.

La programmation par contraintes va utiliser pour chaque sous-problème une méthode de résolution spécifique à ce sous-problème, afin de supprimer les valeurs des domaines des variables impliquées dans le sous-problème qui, compte tenu des valeurs des autres domaines, ne peuvent appartenir à aucune solution de ce sous-problème. Ce mécanisme est appelé filtrage. En procédant ainsi pour chaque sous-problème, donc pour chaque contrainte, les domaines des variables vont se réduire.

Après chaque modification du domaine d'une variable il est utile de réétudier l'ensemble des contraintes impliquant cette variable car cette modification peut conduire à de nouvelles déductions. Autrement dit, la réduction du domaine d'une variable peut permettre de déduire que certaines valeurs d'autres variables n'appartiennent pas à une solution. Ce mécanisme est appelé propagation.

Ensuite et afin de parvenir à une solution, l'espace de recherche va être parcouru en essayant d'affecter successivement une valeur à toutes les variables. Les mécanismes de filtrage et de propagation étant bien entendu relancés après chaque essai puisqu'il y a modification de domaines. Parfois, une affectation peut entraîner la disparition de toutes les valeurs d'un domaine : on dit alors qu'un échec se pro-

duit ; le dernier choix d'affectation est alors remis en cause, il y a "backtrack" ou "retour en arrière" et une nouvelle affectation est tentée.

La programmation par contraintes est donc basée sur trois principes : filtrage, propagation et recherche de solutions.

La programmation par contraintes est d'une grande souplesse puisque l'on peut intervenir à plusieurs niveaux lors de la résolution d'un problème, notamment en :

- définissant de nouvelles contraintes. Il faut donner alors un algorithme permettant de déterminer si la contrainte admet une solution pour un ensemble de domaines quelconque. Dans le meilleur des cas, l'algorithme de filtrage sera directement fourni (il s'agit alors de supprimer les valeurs qui n'appartiennent pas à une solution de la contrainte). Ainsi n'importe quel algorithme de recherche opérationnelle pourra être utilisé en PPC. Cependant, une utilisation efficace demandera une adaptation de l'algorithme en PPC. De nombreux outils de programmation par contraintes proposent des contraintes prédéfinies encapsulant de tels algorithmes.
- définissant des stratégies de recherche de solutions. Il s'agit de définir des critères permettant de choisir la prochaine variable et la prochaine valeur qui sera affectée à cette variable. Afin de mieux comprendre l'intérêt de cet aspect, on peut remarquer qu'un algorithme glouton correspond en fait à une stratégie qui n'est jamais remise en cause. D'une certaine manière on peut donc voir la PPC comme une généralisation des algorithmes gloutons pour les cas où l'on n'est pas certain de la validité de la stratégie gloutonne.[1]

La PPC est très souple puisqu'aucune hypothèse n'est faite ni sur le type des contraintes utilisées ni sur les domaines des variables. Par ailleurs, aucune solution ne peut être perdue puisque toutes les éventualités seront envisagées, c'est pourquoi on parle également d'algorithme énumératif.

Dans la suite, nous verrons que parmi les principes de la PPC que nous venons de présenter certains peuvent être relâchés. Par exemple, il n'est pas nécessaire de supprimer toutes les valeurs qui ne satisfont pas une contrainte, on peut aussi chercher à obtenir plus de filtrage en étudiant a priori les conséquences de l'affectation de chaque valeur à chaque variable pour l'ensemble des contraintes.

1.1 Domaines d'application de la PPC

La programmation par contrainte a pour ambition de résoudre n'importe quel type de problèmes combinatoires aussi elle a été utilisée pour une très grande variété d'applications réelles. Ainsi, on trouve un large éventail d'applications résolues à l'aide d'ILOG Solver sur le site ILOG (www.ilog.fr) que nous reproduisons partiellement ici. Nous avons regroupé par thèmes certaines applications trouvée sur ce site :

Planification et Gestion Nous pouvons citer la planification de la logistique, de la distribution, du personnel, de l'affectation d'équipes, de missions, des techniciens d'installation et de maintenance, de missions satellitaires et de réseaux (dimensionnement et modélisation). ILOG Solver a également été utilisé pour la gestion de la chaîne logistique et d'entrepôts, la configuration et diagnostic d'équipements, l'ordonnancement de lignes de production, l'affectation de fréquences et de bande passante et l'optimisation de charge.

[1] Plus généralement, la stratégie de résolution peut consister à réduire le domaine d'une variable ou à ajouter des contraintes à chaque étape. Le cas où l'on sélectionne une variable et une valeur correspond à l'ajout d'une contrainte du type "variable = valeur".

Transport Dans le domaine des transports, la PPC a fait ses preuves pour des applications telles que l'affectation d'équipages, de comptoirs, de portes d'embarquement et de tapis à bagages, l'ordonnancement d'équipements, la gestion de flottes et la planification du trafic.

Commerce en ligne La PPC est utilisé pour résoudre des applications en ligne aussi variées que la gestion des commandes et des approvisionnements, le service de voyages, la gestion des crédits ou de conseils financiers.

Production industrielle Chrysler a développé un système de planification de la production de ses véhicules intégrant les composants ILOG de PPC. L'application de planification gère la séquence des opérations de peinture des véhicules et améliore la productivité de 15 usines du groupe en Amérique du nord, au Mexique et en Europe. Ce système a déjà permis au producteur automobile de réduire ses coûts de production de 500 000 dollars par an et par usine, soit une économie globale de 7 à 9 millions de dollars par an.

Défense En Grande-Bretagne, la formation des 85 000 militaires de la British Army est planifiée grâce à ILOG Solver.

Dans chacun de ses domaines applicatifs, la PPC est utilisé par des acteurs de renom tels que SAP , Oracle, Daimler-Chrysler, Nissan, Peugeot-Citron (Production Industrielle) ; Siebel et Intershop (Commerce Electronique) ; Sabre et SNCF (Transport) ; Airbus, Lockheed Martin et l'OTAN (Aéronautique, Espace, Défense) ; Deutsche Bank.

1.2 PPC et Recherche Opérationnelle

Nous classons parmi les méthodes de **recherche opérationnelle (RO)**, celles faisant appel aux concepts et aux outils :

- de la programmation mathématique (dont la programmation linéaire, non linéaire et en nombres entiers, et l'optimisation de réseaux)
- des métaheuristiques (notamment, les méthodes de recherche locale, comme la méthode de recherche avec tabous et le recuit simulé, et les méthodes à base de populations, comme les algorithmes génétiques)

De nombreuses méthodes de RO s'appuient sur une appréhension "globale" du problème. Elles travaillent sur une relaxation globale du problème, en général en ne considérant pas certaines contraintes (comme l'intégrité de certaines variables), puis petit à petit essaient de se rapprocher du problème initial, en réintroduisant les contraintes ignorées. A l'inverse la PPC propose de considérer des sous-problèmes (c'est-à-dire des contraintes) et de faire communiquer entre eux ces sous-problèmes (cf. mécanisme de propagation). La PPC a donc une vue beaucoup plus locale, mais exacte, et espère que la propagation apportera un point de vue global. Un des avantages des méthodes de PPC est qu'elles permettent à chaque contrainte de jouer un rôle dans la résolution du problème. C'est pourquoi la méthode présente une grande flexibilité, ce qui constitue l'un de ses principaux attraits. A l'opposé, en dépit de leur efficacité, les méthodes de RO permettent rarement une intégration simple de contraintes additionnelles. Ainsi, il convient d'attirer l'attention sur les problèmes que peuvent poser la recherche de flexibilité : le fait d'ajouter de nouvelles contraintes au problème est relativement facile en PPC, bien qu'il faille exploiter toutes sortes de contraintes. En revanche, réussir à les intégrer dans une méthode de RO sans avoir à tout refaire est plus complexe.

Depuis une dizaine d'années, de nombreux chercheurs combinent des techniques efficaces de recherche opérationnelle (RO), avec les outils flexibles de programmation par contraintes (PPC). On peut ainsi espérer créer des outils d'optimisation à la fois efficaces et conviviaux, même pour des utilisateurs non-spécialistes de l'optimisation combinatoire.

La combinaison entre RO et PPC est souvent qualifiée d'hybridation. Or, sous ce terme, sont présentés différents types de stratégies. Ainsi, nous qualifierons de **coopération** les approches de résolution utilisant la PPC et la RO séparément, par exemple la PPC pour obtenir des stratégies de séparation dans l'arbre de recherche et la RO pour obtenir des bornes en chaque nœud (sans que ces bornes ne servent aux stratégies de séparation). Le terme d'**hybridation** sera utilisé pour caractériser les processus utilisant les deux méthodes pour le même sous-problème, par exemple la PPC pour propager les contraintes issues de coupes obtenues par RO. Enfin, on parlera d'**intégration** de RO en PPC lorsque certains algorithmes internes de PPC utilisent des algorithmes de RO. C'est bien souvent le cas pour les algorithmes de filtrage.

Chapitre 2

Principes de la PPC et problématique de la modélisation

2.1 Principes de la PPC

Nous proposons dans ce chapitre une étude plus technique des notions de base de la PPC.

La programmation par contraintes s'articule autour de quatre entités majeures :

- le réseau de contraintes (CN).
- les algorithmes de filtrage.
- un mécanisme de propagation
- un mécanisme de recherche de solutions, c'est-à-dire de parcours de l'espace de recherche.

2.1.1 Réseau de contraintes

Nous nous limiterons aux réseaux de contraintes à domaines finis.

Un réseau de contraintes fini $\mathcal{N}$ est défini par :

• un ensemble de variables $X = \{x_1, \ldots, x_n\}$,

• un ensemble de domaines $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ où $D(x_i)$ est l'ensemble fini des valeurs possibles pour la variable x_i,

• un ensemble de contraintes $\mathcal{C}$ entre variables. Une contrainte définit les combinaisons de valeurs des variables autorisées.

2.1.2 Algorithme de filtrage

Un algorithme de filtrage, encore appelé algorithme de réduction de domaines, est associé à chaque contrainte. Son rôle est de supprimer des valeurs des domaines des variables de la contrainte pour lesquelles il n'est pas possible de satisfaire la contrainte. Par exemple, pour la contrainte $(x < y)$ avec $D(x) = [10, 20]$, $D(y) = [0, 15]$, un algorithme de filtrage associé à cette contrainte pourra supprimer les valeurs de 15 à 20 de $D(x)$ et les valeurs de 0 à 10 de $D(y)$.

Une des propriétés les plus intéressante d'un algorithme de filtrage est la consistance d'arc. Un algorithme de filtrage associé à une contrainte réalise la consistance d'arc si il supprime toutes les valeurs des variables impliquées dans la contrainte qui ne sont pas consistantes avec la contrainte. Par exemple, pour la contrainte $x+3 = y$ avec les domaines $D(x) = \{1, 3, 4, 5\}$ et $D(y) = \{4, 5, 8\}$, un algorithme de filtrage

établissant la consistance d'arc modifiera les domaines pour obtenir $D(x) = \{1, 5\}$ et $D(y) = \{4, 8\}$.

2.1.3 Mécanisme de propagation

Dès lors qu'un algorithme de filtrage associé à une contrainte modifie le domaine d'une variable, les conséquences de cette modification sont étudiées pour les autres contraintes impliquant cette variables. Autrement dit, les algorithmes de filtrage des autres contraintes sont appelés afin de déduire éventuellement d'autres suppressions. On dit alors qu'une modification a été propagée. Ce mécanisme de propagation est répété jusqu'à ce que plus aucune modification n'apparaisse. Comme les domaines sont finis et comme un algorithme de filtrage est appelé au plus une fois pour chaque modification ce processus se termine nécessairement.

L'idée sous-jacente à ce mécanisme est d'essayer d'obtenir des déductions globales. En effet, on espère que la conjonction des déductions obtenues pour chaque contrainte prise indépendamment conduira à un enchaînement de déductions. C'est-à-dire que cette conjonction est plus forte que l'union des déductions obtenues indépendamment les unes des autres.

2.1.4 Mécanisme de recherche de solutions

Historiquement, le modèle théorique de la PPC, et plus particulièrement des CSP (Constraint Satisfaction Problem), avait pour but de résoudre des problèmes de satisfaction. Aussi, une solution est considérée comme étant une instantiation des variables qui satisfait toutes les contraintes. Lors de la résolution de problèmes d'optimisation, on distinguera deux types de "solutions" : les solutions du problème de satisfaisabilité sous-jacent, c'est-à-dire celles qui ne tiennent pas compte du coût, et les solutions optimales, c'est-à-dire celles qui minimisent (ou maximisent) la fonction de coût.

Le mécanisme de recherche de solutions a pour but de trouver une solution, éventuellement optimale. Il met en œuvre les différents moyens qui vont permettre au solver d'atteindre des solutions. Parmi ces moyens nous pouvons citer :

- les stratégies de choix de variables et de valeurs. Elles définissent les critères qui vont permettre de déterminer la prochaine variable qui sera instanciée ainsi que la valeur qui lui sera affectée.
- les méthodes de décomposition. Lorsque le problème est trop gros, il est souvent nécessairement de le décomposer en plusieurs parties, puis de résoudre ces parties de façon plus ou moins indépendante et enfin de les recombiner.
- les améliorations itératives. Il est souvent illusoire de vouloir trouver et prouver l'optimalité d'un problème de grande taille. Aussi, bien souvent, on recherche quelques "bonnes" solutions puis on essaie de les améliorer à l'aide de techniques d'améliorations locales.

2.2 Modélisation

Dans cette section, nous présentons la problématique de la modélisation. Puis, nous nous attardons sur le concept de contraintes globales qui est majeur en PPC, car ces contraintes contiennent beaucoup plus d'informations. Enfin, nous donnons un exemple de modélisation efficace.

Pour résoudre un problème à l'aide d'un solver, un utilisateur doit définir le réseau de contraintes qu'il considère ainsi que les méthodes de résolutions et les stratégies de choix de variables et de valeurs.

La modélisation d'un problème se fait donc par l'identification de sous-problèmes aisés à résoudre qui vont correspondre aux contraintes choisies.

Trois types de contraintes sont à la disposition de l'utilisateur :

• contraintes prédéfinies du solver (e.g. contraintes arithmétiques, de cardinalité, ...)

• contraintes données en extension, autrement dit par l'ensemble des combinaisons autorisées ou bien interdites

• les contraintes correspondant à des combinaisons de contraintes utilisant les opérateur logiques ET, OU, XOR, NOT. Elles sont parfois appelées meta-contraintes.

En outre, l'utilisateur peut définir ses propres contraintes, en établissant la sémantique de la contrainte, ainsi que l'algorithme de filtrage associé.

Une contrainte peut également être vue comme la recherche de conditions nécessaires vérifiées par toute solution. L'algorithme de filtrage associée à la contrainte se charge alors de supprimer du domaine des variables les éléments qui ne satisfont pas la condition.

L'une des difficultés majeures de la PPC et notamment de la modélisation est l'identification des contraintes. Pour que la résolution ait une chance d'être efficace, deux conditions doivent généralement être remplies :

(i) les contraintes doivent être fortes afin d'engendrer des modifications des domaines des variables.

(ii) les modifications dues à un filtrage doivent pouvoir être utilisées par les autres contraintes.

Ces deux points méritent d'être un peu détaillés.

(i) Le risque de la modélisation est de représenter le problème initial soit comme un ensemble de sous-problèmes très locaux, c'est-à-dire que le problème initial est trop décomposé, soit à l'aide de sous-problèmes correspondant à des relaxations trop fortes du problème réel. Dans le premier cas, les contraintes expriment des conditions très locales et donc trop indépendantes du problème initial. Dans le dernier cas, les contraintes correspondent à des conditions globales mais trop éloignées du problème. Dans les deux cas, les déductions se produiront beaucoup trop tardivement pendant la recherche, alors que l'idéal est que les algorithmes de filtrage associés aux contraintes déduisent rapidement des incohérences afin d'éviter d'étudier des parties importantes de l'espace de recherche.

(ii) Certaines contraintes sont incapables de tirer partie de la déduction faite par d'autres contraintes. Ainsi, après initialisation, la contrainte $(x < y)$ ne peut déduire quelque chose que lors des modifications des bornes de x ou de y. Cela signifie que si le filtrage associé à une contrainte supprime une valeur de x qui n'est ni la valeur minimale de $D(x)$, ni la valeur maximale de $D(y)$ alors le filtrage associé à $(x < y)$ ne fera aucune déduction.

2.2.1 Contraintes globales

Comme la PPC est basée sur des algorithmes de filtrage, il est particulièrement important de définir des algorithmes efficaces et puissants. Aussi, ce thème a attiré l'attention de nombreux chercheurs, qui ont découvert de nombreux algorithmes. Néanmoins, de nombreuses études sur la consistance d'arc se sont limitées aux contraintes binaires définies en extension, c'est-à-dire par la liste des combinaisons de valeurs autorisées. Cette limitation peut être justifiée par le fait que n'importe quelle contrainte peut toujours être définie en extension et par le fait que tout réseau de contraintes non binaires peut être transformé en un réseau équivalent n'impliquant que des contraintes binaires et un certain nombre de variables additionnelles [Rossi et al., 1990]. Cependant, en pratique, cette approche a de nombreux défauts :

– il est souvent inconcevable de transformer une contrainte non binaire en un ensemble de contraintes binaires à cause du coût de traitement d'une telle

opération et de la mémoire requise (particulièrement pour les contraintes dites "non représentables" cf. [Montanari, 1974]).
- la structure des contraintes n'est absolument pas exploitée. Cela empêche le développement d'algorithmes de filtrage efficaces dédiés aux contraintes. De plus, de nombreuses contraintes non binaires perdent totalement leurs structures lorsqu'elles sont représentées par un ensemble de contraintes binaires. Cela conduit, par exemple, à moins de filtrage de la part des algorithmes de filtrage par consistance d'arc associés à ces contraintes. En effet, deux réseaux de contraintes équivalents en terme de solutions, n'auront pas nécessairement les mêmes domaines après fermeture par consistance d'arc de chaque contraintes.

L'intérêt de l'utilisation de la structure des contraintes peut être mis en évidence à l'aide d'un exemple simple : la contrainte $x \leq y$. Soient $min(D)$ et $max(D)$ respectivement la valeur minimum et la valeur maximum d'un domaine D. Il est évident que toutes les valeurs de x et de y de l'intervalle $[min(D(x)), max(D(y)]$ sont consistantes avec la contrainte. Cela signifie que la consistance d'arc peut être facilement et efficacement réalisée en supprimant toutes les valeurs qui ne sont pas dans l'intervalle ci-dessus. Par ailleurs, l'utilisation de la structure des contraintes est souvent la seule manière d'éviter les problèmes de consommation mémoire liés aux contraintes non binaires. En fait, cette approche évite de représenter explicitement toutes les combinaisons de valeurs autorisées par la contrainte.

En conséquence, les chercheurs intéressés par la résolution d'applications réelles avec la PPC, et particulièrement ceux qui développent des langages de PPC (comme PROLOG), ont écrit des algorithmes de filtrage spécifiques aux contraintes simples les plus communes (comme $=, \neq, <, \leq$, ...) ainsi que des cadres formels généraux permettant d'exploiter efficacement certaines structures des contraintes binaires (comme AC-5 [Van Hentenryck et al., 1992]). Les chercheurs ont alors été confrontés à deux nouveaux problèmes : le manque d'expressivité de ces contraintes simples et la faiblesse des réductions de domaines entraînés par les algorithmes de filtrage associés à ces contraintes.

Intéressons nous tout d'abord à l'expressivité. Il est, en effet, beaucoup plus agréable pour modéliser un problème en PPC d'avoir à sa disposition des contraintes correspondant à un ensemble de contraintes simples. Ces contraintes encapsulant un ensemble d'autres contraintes sont appelées **contraintes globales**. Formellement, on a :

Définition 1 *Soit $\mathcal{C} = \{C_1, C_2, .., C_n\}$ un ensemble de contraintes. La contrainte C_G égale à la conjonction de toutes les contraintes de $\mathcal{C}$: $C_G = \wedge\{C_1, C_2, ..C_n\}$ est une* **contrainte globale**.
L'ensemble de tuples de $\mathcal{C}$ est égal à l'ensemble de solutions de
$(\cup_{C \in \mathcal{C}} X(C), \mathcal{D}_{X(\mathcal{C})}, \{C_1, C_2, .., C_n\})$.

Par exemple, une contrainte alldiff définie sur un ensemble X de variables impose que les valeurs prises par ces variables soient deux à deux différentes. Il est beaucoup plus simple de définir une seule contrainte alldiff(X), plutôt que de définir une contrainte de différence entre chaque paire de variables de X.

De plus, ces nouvelles contraintes peuvent être associées à des algorithmes de filtrage beaucoup plus puissants parce qu'elles peuvent prendre en compte simultanément la présence de plusieurs contraintes simples afin de réduire plus le domaine des variables. Nous pouvons mettre en évidence cet aspect avec un exemple plus réaliste qui implique des contraintes globales de cardinalité (GCC).

Une GCC est définie par un ensemble de variables $X = \{x_1, ..., x_p\}$ qui prennent leurs valeurs dans un ensemble $V = \{v_1, ..., v_d\}$. Elle contraint le nombre de fois où chaque valeur $v_i \in V$ est affecté à une variable de X à appartenir à un intervalle $[l_i, u_i]$. Les GCC apparaissent dans de nombreux problèmes réels. Considérons

	Mo	Tu	We	Th	...
peter	D	N	O	M	
paul	D	B	M	N	
mary	N	O	D	D	
...					

A = {M,D,N,B,O}, P = {peter, paul, mary, ...}
W = {Mo, Tu, We, Th, ...}
M : morning, D : day, N : night B : backup, O : day-off

FIG. 2.1 – Un problème d'emploi du temps.

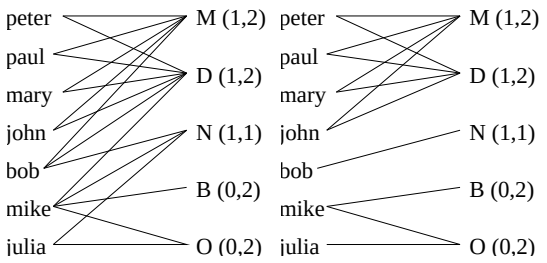

FIG. 2.2 – Un exemple de contrainte globale de cardinalité.

un exemple dérivé d'un problème réel et présenté par [Caseau et al., 1993] (cf. Figure 2.1). Le but est de définir l'emploi du temps de managers d'un centre d'assistance comportant 5 activités, représentés par l'ensemble A, 7 personnes, représentées par l'ensemble P pour une période de 7 jours, représentée par l'ensemble W. Chaque jour, une personne doit effectuer une des activités de l'ensemble A. Le but est de définir une matrice d'affectation qui satisfait les contraintes générales et locales suivantes :

- Les **contraintes générales** restreignent les affectations. Pour chaque jour, chaque activité doit être affectée un certain nombre de fois compris entre un minimum et un maximum donnés. Pour toute période de 7 jours, une personne doit effectuer un certain nombre de fois chaque activité. Aussi, pour chaque ligne et pour chaque colonne de la matrice, une contrainte de cardinalité globale est définie.
- Les **contraintes locales** indiquent principalement des incompatibilités entre deux jours consécutifs. Par exemple, on ne peut pas travailler le matin après avoir travaillé la nuit précédente.

Chaque contrainte générale peut être représentée par autant de contraintes min/max qu'il y a d'activités. Ces contraintes min/max peuvent être facilement manipulées grâce, par exemple, aux opérateurs **atmost/atleast** proposés par [Van Hentenryck and Deville, 1991]. De tels opérateurs sont implémentés sous forme d'algorithmes de filtrage locaux. Or, il a été remarqué dans [Caseau et al., 1993] : "Le problème est qu'une résolution efficace de problèmes d'emploi du temps requiert un calcul global pour l'ensemble des contraintes min/max, et non pas une implémentation efficace de chacune d'elles séparément". C'est pourquoi, cette manière de procéder n'est pas satisfaisante. Aussi, les contraintes globales de cardinalité associées à des algorithmes de filtrage efficaces (comme ceux réalisant la consistance d'arc) sont nécessaires.

Afin de montrer les différences entre un filtrage global et un ensemble de filtrage locaux, nous considérons une GCC associée à une journée (voir figure 2.2). Cette contrainte peut être représentée par un graphe biparti (graphe de gauche dans la

figure 2.2). L'ensemble gauche correspond à l'ensemble des personnes et l'ensemble droit à l'ensemble des activités. Il existe une arête entre une personne et une activité lorsque la personne peut être affectée à l'activité. Pour chaque activité les nombres entre parenthèses indiquent le nombre minimum et maximum de personnes qui peuvent être affectées à l'activité. Par exemple, John peut travailler le matin et la journée, mais pas la nuit ; au moins un manager doit travailler le matin, et au plus deux managers peuvent travailler le matin. Nous rappelons que chaque personne doit être affectée à une et une seule activité.

Modéliser le problème avec un ensemble de contraintes atmost/atleast ne conduit à aucune suppression de valeur. En effet, une contrainte $atleast(X, \#time, val)$ est une contrainte locale. Si une telle contrainte est considérée individuellement alors la valeur val ne peut pas être supprimée d'un domaine tant qu'elle appartient plus de $\#time$ fois aux domaines des variables de X. Un algorithme de filtrage par consistance d'arc pour cette contrainte affectera une variable x à val si et seulement si il ne reste plus exactement que $\#time$ variables dont le domaine contient val. De façon similaire, si une contrainte $atmost(X, \#time, val)$ est considérée individuellement alors la valeur val ne peut pas être supprimée d'un domaine tant que $\#time$ variables n'ont pas été affectées à cette valeur. Un algorithme de filtrage par consistance d'arc pour cette contrainte supprimera val du domaine d'une variable x si et seulement si $\#time$ variables différentes de x sont affectées à val. On remarquera qu'aucun de ces cas ne se produit pour l'exemple considéré.

Or, avec une étude particulière de la contrainte on peut déduire certaines choses. Peter, Paul, Mary, et John peuvent travailler uniquement le matin ou la journée. De plus, le matin et la journée ne peuvent être affectés au plus qu'à quatre personnes, donc, aucune autre personne (i.e. Bob, Mike, ou Julia) ne peuvent effectuer les activités M et D. Aussi, nous pouvons supprimer les arêtes entre Bob, Mike, Julia et D, M, autrement dit éliminer les valeurs D et M des variables correspondant aux personnes Bob, Mike et Julia. Maintenant, il n'y a plus qu'une seule possibilité pour Bob : N, qui ne peut être affecté qu'au plus une fois. C'est pourquoi, nous pouvons supprimer les arêtes {mike,N} et {julia,N}. Ce raisonnement conduit au graphe de droite de la Figure 2.2. Il correspond à la réalisation de la consistance d'arc pour la contrainte globale de cardinalité.

Le filtrage est une propriété locale. Si on décompose une contrainte, on obtient alors un ensemble de filtrages plus faibles car moins informé. Nous pouvons formellement mettre en évidence ces différences entre filtrages par la propriété suivante :

Propriété 1 *La réalisation de la consistance d'arc pour une contrainte* $\mathcal{C} = \wedge\{C_1, C_2, .., C_n\}$ *est plus forte (autrement dit ne peut pas supprimer moins de valeurs) que la réalisation de la consistance d'arc du réseau* $(\cup_{C\in\mathcal{C}} X(C), \mathcal{D}_{X(\mathcal{C})}, \{C_1, C_2, .., C_n\})$.

preuve : D'après la définition 1 l'ensemble des tuples de $\mathcal{C} = \wedge\{C_1, C_2, .., C_n\}$ correspond à l'ensemble des solutions du réseau $(\cup_{C\in\mathcal{C}} X(C), \mathcal{D}_{X(\mathcal{C})}, \{C_1, C_2, .., C_n\})$. Donc, la réalisation de la consistance d'arc pour $\wedge\{C_1, C_2, .., C_n\}$ supprime toutes les valeurs qui n'appartiennent pas à une solution de $(\cup_{C\in\mathcal{C}} X(C), \mathcal{D}_{X(\mathcal{C})}, \{C_1, C_2, .., C_n\})$ ce qui est plus fort que réaliser la consistance d'arc sur ce réseau.

Aussi la consistance d'arc pour une contrainte globale est une propriété forte. La proposition suivante montre cette force en exhibant un réseau de contrainte pour lequel la consistance d'arc est équivalente à celle d'une contrainte alldiff.

Proposition 1 *La consistance d'arc pour* $C =$*alldiff*(X) *correspond à la consistance d'arc pour le réseau de contrainte ayant un nombre exponentiel de contraintes défini par :*
$\forall A \subseteq X : |D(A)| = |A| \Rightarrow D(X - A)$ *est réduit à* $D(X) - D(A)$

preuve : voir [Régin, 1995].

Enfin, remarquons qu'il est également possible de définir des algorithmes de filtrage simple pour les contraintes globales juste en prenant en compte la présence simultanée de contraintes.

Considérons, par exemple un ensemble de 5 variables : $X = \{x_1, x_2, x_3, x_4, x_5\}$ dont les domaines sont $D(x_1) = 0$, $D(x_2) = 0$, $D(x_3) = 0$, $D(x_4) = 0, 1, 2, 3, 4$, $D(x_5) = 0, 1, 2, 3, 4$; et quatre contraintes $atleast(X, 1, 1)$, $atleast(X, 1, 2)$, $atleast(X, 1, 3)$, et $atleast(X, 1, 4)$ ce qui signifie que chaque valeur de $\{1, 2, 3, 4\}$ doit être prise au moins une fois par une variable de X dans toute solution.

Il est clair que le problème considéré n'a pas de solutions, car quatre valeurs doivent être prise au moins une fois et il n'existe que deux variables pouvant les prendre. Or, si l'on considère les filtrages associés aux contraintes atmost et atleast prises individuellement, comme nous l'avons présenté précédemment, on s'aperçoit qu'aucune valeur n'est supprimée d'un domaine. En effet, les domaines des variables x_4 et x_5 restent les mêmes parce que toute valeur de $\{1, 2, 3, 4\}$ appartient à au moins deux domaines.

Pour cet exemple, nous pouvons déduire une nouvelle contrainte à partir de la présence simultanée de plusieurs contraintes. Si quatre valeurs doivent être prises au moins une fois par cinq variables alors les autres valeurs ne peuvent être prises qu'au plus $5 - 4 = 1$ fois, donc nous pouvons ajouter la contrainte $atmost(x, 1, 0)$. En introduisant cette nouvelle contrainte un échec est immédiatement déduit.

Cette idée peut être généralisée pour n'importe quelle contrainte globale de cardinalité. Soit $card(a_i)$ la variable associée à chaque valeur a_i de $D(X)$ qui compte le nombre de domaines de X qui contiennent a_i. Nous avons $l_i \leq card(a_i) \leq u_i$, où l_i et u_i sont respectivement le minimum et le maximum de fois où la valeur a_i doit être prise. Alors, nous pouvons simplement déduire la contrainte $\sum_{a_i \in D(X)} card(a_i) = |X|$; et chaque fois que le minimum ou le maximum de $card(a_i)$ sont modifiés, les valeurs de l_i et u_i sont mises à jour et la GCC est modifiée.

Cette méthode montre comment on peut définir simplement un filtrage plus puissant en introduisant des contraintes supplémentaire. Bien entendu, la consistance d'arc de la contrainte globale n'est pas assurée, mais la méthode est simple et facile à mettre en œuvre.

Pour résumer, nous pouvons donc énumérer trois intérêts majeurs pour les contraintes globales :

- L'expressivité : il est plus pratique de définir une contrainte correspondant à un ensemble de contraintes plutôt que de définir indépendamment chacune des contraintes de cet ensemble.
- Comme une contrainte globale correspond à un ensemble de contraintes il est possible de déduire plus d'informations à partir de la présence simultanée de contraintes.
- Des algorithmes de filtrage puissants prenant en compte l'ensemble des contraintes comme un tout peuvent être écrits. Ces algorithmes de filtrage rendent possible l'utilisations de techniques de Recherche Opérationnelle ou de théorie des graphes en PPC.

De nombreuses contraintes globales ont ainsi été développées. Nous pouvons citer par exemple la contrainte cumulative [Beldiceanu and Contejean, 1994] [Beldiceanu and Carlsson, 2002], la contrainte d'ordonnancement d'activités non interruptibles [Carlier and Pinson, 1994, Baptiste et al., 1998], la contrainte diffn [Beldiceanu et al., 2001], la contrainte cycle [Beldiceanu and Contejean, 1994], la contrainte sort [Zhou, 1996, Zhou, 1997] [Bleuzen-Guernalec and Colmerauer, 1997], la contrainte alldiff [Régin, 1994], la contrainte symmetric alldiff [Régin, 1999b], la contrainte globale de cardinalité [Régin, 1996], la contrainte globale de cardinalité

avec coûts [Régin, 1999a, Régin, 2002], la contrainte de produit scalaire de variables toutes différentes [Régin, 1999a], la contrainte séquence [Régin and Puget, 1997], la contrainte stretch [Pesant, 2001], la contrainte globale de distance minimum [Régin, 1997], la contrainte k-diff [Régin, 1995] et la contrainte du nombre de valeurs distinctes [Beldiceanu, 2001a]

Enfin, mentionnons l'état de l'art sur l'usage de contraintes globales de H. Simonis [Simonis, 1996].

2.2.2 Un exemple de modélisation efficace

Nous proposons dans cette section de donner brièvement un modèle efficace pour résoudre un problème difficile de calcul de calendrier de tournois sportifs décrit dans [McAloon et al., 1997] et [Van Hentenryck et al., 1999].

Le problème consiste à déterminer les matchs entre n équipes pour $n-1$ semaines, en sachant que chaque semaine est divisée en $n/2$ périodes. Le but est donc de définir pour chaque période et pour chaque semaine le match qui doit être joué en tenant compte des contraintes suivantes :

1. Chaque équipe doit jouer contre toutes les autres équipes.
2. Une équipe joue exactement une fois chaque semaine
3. Une équipe ne peut jouer qu'au plus deux fois pendant la saison dans la même période.

La table suivante donne une solution du problème pour 8 équipes :

	Sem. 1	Sem. 2	Sem. 3	Sem. 4	Sem. 5	Sem. 6	Sem. 7
Period 1	1 vs 2	1 vs 3	4 vs 8	4 vs 7	4 vs 8	2 vs 6	3 vs 5
Period 2	3 vs 4	2 vs 8	1 vs 4	6 vs 8	2 vs 5	1 vs 7	6 vs 7
Period 3	5 vs 6	4 vs 6	2 vs 7	1 vs 5	3 vs 7	3 vs 8	1 vs 8
Period 4	7 vs 8	5 vs 7	3 vs 6	2 vs 3	1 vs 6	4 vs 5	2 vs 4

Ce problème est particulièrement intéressant pour la PPC. Tout d'abord parce que c'est un benchmark standard (proposé par Bob Daniel) des problèmes MIP et il est affirmé (cf. [McAloon et al., 1997]) que les meilleurs solvers MIP ne peuvent pas trouver une solution pour 14 équipes, alors que le modèle proposé dans cette section est nettement plus efficace. Ensuite, cet exemple met en évidence les caractéristiques fondamentales de la PPC, autrement dit l'utilisation de contraintes globales. En particulier cet exemple utilise la consistance d'arc des contraintes globales de cardinalité.

L'idée principale est d'utiliser deux classes de variables : pour une semaine et une période données, les variables d'équipes spécifient l'équipe qui joue et les variables de matchs expriment le match qui est joué. L'utilisation de variables de matchs rend plus simple l'expression de la contrainte imposant que toutes les équipes doivent se rencontrer une et une seule fois. Les matchs sont identifiés sans ambiguité à l'aide des deux équipes qui le compose. Plus précisément, un match formé par l'équipe h jouant à domicile et l'équipe a jouant à l'extérieur est identifié par l'entier $h * n + a$.

On peut poser une contrainte entre les deux équipes jouant ensemble afin de casser une symétrie. En effet, pour chaque période et pour chaque semaine données le problème ne différencie pas le match (a vs b) du match (b vs a). On peut donc imposer la contrainte établissant que seul le match (a vs b) avec $a < b$ doit être envisagé.

Les variables d'équipes et les variables de matchs doivent être liées ensemble afin de s'assurer que pour une période et une semaine données les valeurs possibles pour la variable de match et celles des deux variables d'équipes sont cohérentes entre elles. Ce lien est mis en œuvre à l'aide d'une contrainte dont l'ensemble des tuples

est donné en extension. Pour 8 équipes, cet ensemble est constitué des tuples de la forme $(1, 2, 1)$ (ce qui signifie que le match opposant les équipes 1 et 2 est le match numéro 1), $(1, 3, 2)$, ..., $(7, 8, 56)$. Donc pour chaque intersection entre une ligne et une colonne, on définit une telle contrainte.

Le problème peut être rendu plus uniforme en introduisant une colonne supplémentaire que l'on nomme "extra" colonne. On peut alors modifier la contrainte sur les lignes imposant qu'une équipe sera présente au plus deux fois par période. En effet, on peut remarquer qu'une équipe doit jouer $n-1$ matchs, puisqu'elle a $n-1$ adversaires possibles. Or, une équipe ne peut jouer qu'au plus deux fois par période et il y a $n/2$ périodes, donc une équipe jouera deux fois par période pour toutes les périodes, sauf une pour laquelle elle ne joue qu'une seule fois. Aussi, pour chaque période il y a toujours exactement deux équipes qui ne jouent qu'une et une seule fois pour cette période. Donc, si l'on place pour chaque période ces deux équipes dans la colonne "extra", on peut changer la contrainte sur les périodes : en incorporant la colonne "extra" chaque équipe doit jouer exactement deux fois par période. En outre, chaque équipe n'est introduite dans la colonne "extra" qu'une et une seule fois, donc pour cette colonne on introduit une nouvelle contrainte alldiff. Ainsi, nous avons pour chaque période une contrainte globale de cardinalité impliquant toutes les variables d'équipes de la période (la colonne "extra" étant incluse) et imposant que toute équipe doit être prise exactement deux fois par ces variables. Pour chaque semaine, nous définissons une contrainte alldiff impliquant toutes les variables d'équipes de cette semaine. On introduit également une contrainte alldiff pour la colonne "extra".

La stratégie de sélection d'affectation consiste à sélectionner l'équipe la plus affectée puis à l'affecter à la variable ayant le moins de valeur possible et contenant l'équipe choisie dans son domaine.

On obtient alors les résultats suivants (les temps sont exprimés en secondes sauf mention particulière) :

#équipes	8	10	12	14	16	18	20	22	24
#échecs	32	417	41	3,514	1,112	8,756	72,095	6,172,672	6,391,470
temps	0.1	0.3	0.1	0.1	1.5	12	110	3h	4h

Pour ce modèle il est indispensable d'utiliser des algorithmes de filtrage puissants, notamment ceux réalisant la consistance d'arc pour les contraintes globales. Un modèle encore plus performant (le problème avec 40 équipes est résolu) est proposé dans [Van Hentenryck et al., 1999]. Cet autre modèle propose d'engendrer d'abord un calendrier puis d'essayer de satisfaire les contraintes de périodes. On s'affranchit ainsi des contraintes de calendrier, autrement dit des contraintes sur les colonnes et de la contrainte imposant que chaque équipe doit rencontrer chaque autre équipe exactement une fois.

2.2.3 Contribution Personnelle

J'ai personnellement proposé les deux modèles les plus efficaces pour résoudre le problème précédent. Ces modèles ont tout d'abord été exposés à la conférence INFORMS en 1998 [Régin, 1998], puis ils ont été repris comme exemple de programme OPL (voir [Van Hentenryck et al., 1999]). Cet exemple a été utilisé à de nombreuses reprises par moi-même ou par d'autres personnes comme Pascal van Hentenryck afin de montrer aux chercheurs de Recherche Opérationnelle l'intérêt de la PPC.

J'ai présenté des travaux généraux sur la modélisation à un workshop DIMACS en 1998. L'article de ce workshop a été publié dans une revue en 2001. J'essaie dans cet article de définir un bon modèle et je donne notamment les concepts de bases à

la création d'un modèle efficace. Cet article est malheureusement peu connu de la communauté.

Enfin, peu de temps après ma thèse, j'ai publié avec Christian Bessière un article qui a eu un retentissement certain dans la communauté [Bessière and Régin, 1996]. Cet article montre qu'il est préférable d'utiliser des filtrages puissants et que les méthodes basées sur les filtrages sont plus efficaces que les méthodes de retour-arrière (backtrack) intelligents. Il faut noter que cela ne faisait que confirmer les dires des personnes utilisant des solvers pour résoudre des applications réelles, comme Pascal van Hentenryck ou Jean-François Puget (ILOG).

Chapitre 3

Algorithmes de filtrage et contraintes globales

3.1 Algorithmes génériques de filtrage

Ecrire un algorithme de filtrage dédié à une contrainte particulière est une tâche difficile et qui demande du temps. Dès lors, il est légitime de se poser la question de l'existence et de l'efficacité d'algorithmes génériques. C'est le propos de cette section.

Nous introduisons une notation particulière $\mathcal{D}_0 = \{D_0(x_1), \ldots, D_0(x_n)\}$ pour représenter l'ensemble des domaines de définition de $\mathcal{N}$. En effet, on peut considérer que n'importe quel réseau de contraintes peut être associé à un ensemble de domaines initiaux $\mathcal{D}_0$ (contenant $\mathcal{D}$), sur lesquels les contraintes sont définies.

Une contrainte C définie sur un ensemble ordonné de variables $X(C)=(x_{i_1}, .., x_{i_r})$ est un ensemble $T(C)$ du produit Cartésien $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ qui spécifie les combinaisons autorisées de valeurs pour les variables $x_{i_1}, \ldots, x_{i_r}$. Un élément de $D_0(x_{i_1}) \times \cdots \times D_0(x_{i_r})$ est appelé un tuple de $X(C)$. $|X(C)|$ est l'arité de C.

Une valeur a d'une variable x est souvent notée (x, a). $var(C, i)$ représente la i^{eme} variable de $X(C)$, alors que $index(C, x)$ est la position de la variable x dans $X(C)$. $\tau[k]$ est la k^{eme} valeur du tuple τ. $\tau[index(C, x)]$ sera notée $\tau[x]$ lorsqu'aucune confusion n'est possible. $D(X)$ est l'union des domaines des variables de X (i.e. $D(X) = \cup_{x_i \in X} D(x_i)$). $\#(a, \tau)$ est le nombre d'occurrences de la valeur a dans le tuple τ.

Soit C une contrainte. Un tuple τ de $X(C)$ est valide si $\forall (x, a) \in \tau, a \in D(x)$. C est consistante si et seulement s'il existe un tuple τ de $T(C)$ qui soit valide. Une valeur $a \in D(x)$ est consistante avec C si et seulement si $x \notin X(C)$ ou s'il existe un tuple valide τ de $T(C)$ avec $a = \tau[x]$ (τ est appelé un support pour (x, a) sur C). Une contrainte est arc consistante si et seulement si $\forall x_i \in X(C), D(x_i) \neq \varnothing$ et $\forall a \in D(x_i)$, a est consistante avec C.

3.1.1 Consistance d'arc binaire

Présentation

Cette partie ne présente pas le détail des algorithmes. Le lecteur plus particulièrement intéressé pourra se référer à [Régin, 2004]. L'article décrivant cet algorithme a deux avantages : il est en français et il contient un état de l'art détaillé de tous les algorithmes publiés jusqu'à maintenant. Nous nous contenterons donc de rappeler l'historique des différents travaux.

Le calcul de la fermeture par arc consistance d'une contrainte binaire est un domaine de la PPC qui a toujours suscité l'intérêt de nombreux chercheurs, notamment ceux de "l'école" CSP. Il s'agit de déterminer les valeurs des variables d'une contrainte binaire donnée en extension qui ne sont pas consistantes avec la contrainte.

L'un des premiers algorithmes a été proposé par A. Mackworth [Mackworth, 1977]. Cet algorithme est très simple conceptuellement et très facile à implémenter, malheureusement il est trop systématique et ne mémorise aucun calcul précédent. Sa complexité en temps est en $O(d^3)$ pour une contrainte (d désigne la taille du plus grand domaine des deux variables). En 1986, Mohr et Henderson ont décrit le premier algorithme optimal : AC-4 [Mohr and Henderson, 1986], optimal signifie que dans le pire des cas la complexité est du même ordre de grandeur que le nombre maximaux de suppressions possibles. En conséquence, AC-4 a une complexité en temps en $O(d^2)$. Cependant, cet algorithme nécessite de nombreux précalculs, ce qui conduit à une complexité en espace en $O(d^2)$ et ce qui entraîne un nombre de calculs en pratique proche de ceux de la complexité dans le pire des cas. On notera néanmoins qu'AC-4 est le premier algorithme incrémental. Afin de remédier aux problèmes de précalculs systématiques et de consommation mémoire importante (autant que de combinaisons de valeurs autorisées) C. Bessière et M-O. Cordier ont modifié AC-4 en introduisant une nouvelle notion majeure : celle de support unique. Ce travail a conduit à l'algorithme AC-6 [Bessière and Cordier, 1993], qui a une complexité en espace réduite à $O(d)$ tout en gardant la même complexité en temps que celle d'AC-4. De plus, AC-6 s'avère nettement plus performant en pratique, bien qu'il soit plus complexe à implémenter.

Résultats

J'ai apporté avec C. Bessière et Gene Freuder différentes améliorations à l'algorithme AC-6. Tout d'abord, AC-7 ([Bessière and Régin, 1994]) propose de prendre en compte ce que l'on appelle la bidirectionnalité des supports, c'est-à-dire que si (x, a) supporte (y, b) alors (y, b) supporte aussi (x, a), tout en conservant les mêmes complexités en temps et en espace. Cette idée à été étendue pour donner naissance aux algorithme AC-Inference et AC-Identical [Bessière et al., 1999]. Ces algorithmes prennent en compte diverses règles d'inférence comme la réflexivité ou la commutativité de certaines contraintes. AC-identical tire parti du fait que certaines contraintes identiques (i.e. ayant le même ensemble de tuples) sont souvent définies plusieurs fois en impliquant des variables différentes dans un problème. Ces dernièrs algorithmes conservent une complexité optimale en temps mais requièrent un espace mémoire en $O(d^2)$.

Bien qu'AC-6 et ses diverses améliorations donnent de bons résultats en pratique, deux problèmes demeuraient. Ces algorithmes sont conceptuellement plus complexes qu'AC-3 et ils ne sont pas comparables à AC-3 : il existe des cas où AC-3 est le meilleur et d'autres pour lesquels c'est AC-6. Aussi, j'ai proposé avec C. Bessière AC-2000 une amélioration simple d'AC-3 et AC-2001 un algorithme basé sur les principes d'AC-3 qui emprunte certaines idées d'AC-6 afin d'éviter de refaire certains calculs. Les complexités d'AC-2001 sont identiques à celle d'AC-6. L'avantage d'AC-2001 réside aussi dans le fait que l'on peut prouver qu'il est toujours meilleur qu'AC-3 et on peut exactement le comparer à AC-6, c'est-à-dire que l'on peut déterminer exactement pour une contrainte donnée et un ensemble de domaines lequel des deux algorithmes sera le meilleur.

Enfin, j'ai proposé CAC, un algorithme générique, configurable et adaptatif. Cet algorithme est une généralisation d'AC-5 [Van Hentenryck et al., 1992] qui permet de représenter n'importe quel algorithme existant et qui peut à tout instant utiliser le meilleur algorithme. Dans cet article une nouvelle nomenclature des algorithmes

d'AC est également proposé.

Travaux connexes

Notons tout d'abord qu'AC-2001 a été découvert en même temps par Y. Zhang et R. Yap [Zhang and Yap, 2001]. Ils ont nommé leur algorithme AC-3.1.

Divers travaux ont été entrepris à propos de l'ordre de propagation des contraintes, la plupart étant basés sur AC3. Ces travaux ont donné naissance à de nouveaux algorithmes. Nous pouvons citer AC-8 [Chmeiss and Jégou, 1998], AC-3_d [van Dongen, 2002].

C. Lecoutre, F. Boussemard et F. Hemery ont proposé d'intégrer certains principes d'AC-7 dans AC-3, cela leur a permis d'obtenir AC-3.2 et AC-3.3 [Lecoutre et al., 2003].

Enfin, M.R.C. van Dongen est certainement l'un des chercheurs les plus actifs du domaine. Il a soutenu sa thèse en 2002 [van Dongen, 2002] et a notamment proposé de rechercher deux supports au lieu d'un seul, d'où l'algorithme AC-3_b [van Dongen, 1997, van Dongen and Bowen, 2000].

3.1.2 Consistance d'arc non binaire

Présentation

En PPC pour résoudre un problème, on commence par définir un modèle en utilisant des contraintes prédéfinies, comme une somme, un alldiff... Ensuite on défini d'autres contraintes spécifiques au problème. Puis, on appelle une procédure de recherche d'une solution.

Souvent, lorsque l'on cherche à résoudre un problème réel, appelons le P, les différents modèles simples testés s'avèrent incapable de résoudre P dans un temps raisonnable. Dans ce cas, nous devons considérer des sous-problèmes de P, par exemple R, et essayer d'améliorer la résolution de R dans l'espoir que cela améliorera aussi la résolution de P. Autrement dit, nous essayons d'identifier des sous-problèmes de P pour lesquels on peut définir une contrainte et un algorithme de filtrage associé. Plus précisément cela revient, pour chaque sous-problème pertinent de P, à définir une contrainte globale qui correspond à la conjonction des contraintes impliquées dans le sous-problème.

Supposons que les algorithmes de filtrage associés à ces contraintes réalisent la consistance d'arc et que cela améliore la résolution de P (par exemple en entrainant une diminution notable du nombre de backtracks). Dans ce cas, la résolution des sous-problèmes R a une forte influence sur la résolution de P, donc il est intéressant d'écrire de tels algorithmes de filtrage. A l'inverse, si l'introduction de tels algorithmes ne modifie que légèrement la résolution de P, par exemple s'ils entraînent une diminution faible du nombre de backtracks, alors nous savons que la résolution des sous-problèmes R a un impact limité sur la résolution de P et donc il n'est pas très intéressant de travailler sur ces algorithmes de filtrage. En procédant de cette manière nous pourrons améliorer la résolution de P. Aussi, un algorithme de filtrage général va s'avérer très intéressant en pratique, car grâce a lui nous pourrons déterminer les sous-problèmes les plus importants. Ensuite, on écrira des algorithmes dédiés, donc plus efficaces, pour chacun de ces sous-problèmes.

Résultats

J'ai écrit avec C. Bessière [Bessière and Régin, 1997] un schéma général, appelé GAC-Schema, qui permet de calculer la consistance d'arc pour des contraintes non binaires. Cet algorithme admet en entrée la liste des tuples de la contrainte (c'est-à-dire la liste des combinaisons autorisées) ou la liste des combinaisons interdites. Ce

dernier point est important car dans le cas non binaire, une contrainte impliquant n variables peut avoir un nombre de tuples en $O(d^n)$, et un nombre polynomial de combinaisons interdites.

Ce schéma général fonctionne aussi avec n'importe quel type de fonction retournant une solution. Nous proposons de donner les idées générales de cet algorithmes dans cette section.

Supposons que l'on dispose d'une fonction, notée existSolution(P), qui soit capable de déterminer si un problème particulier $P = (X, \mathcal{C}, \mathcal{D})$ a une solution ou non. Considérons alors la contrainte globale $C(P)$ qui encapsule le problème P (i.e. $C(P)$ correspond à l'ensemble des contraintes de P).

Par souci de clarté nous noterons $P_{x=a}$ le problème P pour lequel on impose que $x = a$, en d'autres termes $P_{x=a} = (X, \mathcal{C} \cup \{x = a\}, \mathcal{D})$.

Etablir la consistance d'arc de $C(P)$ se fait en recherchant des supports pour les valeurs des variables sur lesquelles $C(P)$ est définie. Un support pour une valeur (y, b) pour $C(P)$ peut être recherché par n'importe quelle procédure de recherche puisqu'un support pour (y, b) est une solution de $P_{y=b}$.

Un premier algorithme

Un premier algorithme très simple consiste à appeler la fonction existSolution avec $P_{x=a}$ comme paramètre pour chaque valeur a de chaque variable x impliquée dans P, et ensuite à supprimer a de x lorsque existSolution($P_{x=a}$) n'a pas de solution. l'algorithme 3.1 est une implémentation possible de cette idée.

```
SimpleGeneralFilteringAlgorithm(C(P), deletionSet) : boolean
for each a ∈ X do
    for each a ∈ D(x) do
        if ¬ existSolution(P_{x=a}) then
            remove a from D(x)
            if D(x) = ∅ then return false
            add (y, b) to deletionSet
return true
```

Algorithm 3.1 – Un premier algorithme générique de filtrage réalisant la consistance d'arc.

Cet algorithme est très simple mais il n'est pas très efficace parce que chaque fois qu'une valeur est supprimée, l'existence de solution pour toutes les autres affectations doit être testée à nouveau.

Si $O(P)$ est la complexité de la fonction existSolution(P) alors nous pouvons récapituler la complexité de cet algorithme dans le tableau suivant :

	test de consistance		consistance d'arc	
	meilleure	pire	meilleure	pire
A partir du début	$\Omega(P)$	$O(P)$	$nd \times \Omega(P)$	$nd \times O(P)$
Après k modifications	$k \times \Omega(P)$	$k \times O(P)$	$knd \times \Omega(P)$	$knd \times O(P)$

Un meilleur algorithme

On peut proposer un algorithme générique bien meilleur à condition que la fonction existSolution(P) retourne une solution lorsqu'il y en a une, au lieu d'un booléen.

Tout d'abord, considérons qu'une valeur (x, a) a été supprimée de $D(x)$. Nous devons étudier les conséquences de la disparition de cette valeur. Donc, pour chaque valeur qui est supportée par un tuple contenant (x, a) un autre support doit être

```
    GeneralFilteringAlgorithm(C(P), x, a, deletionSet) : boolean
1   for each τ ∈ S_C(x, a) do
        for each (z, c) ∈ τ do remove τ from S_C(z, c)
2       for each (y, b) ∈ S(τ) do
            remove (y, b) from S(τ)
            if b ∈ D(y) then
3               σ ← seekInferableSupport(y, b)
                if σ ≠ nil then add (y, b) to S(σ)
                else
4                   σ ← existSolution(P_{y=b})
                    if σ ≠ nil then
                        add (y, b) to S(σ)
                        for k = 1 to |X(C)| do add σ to S_C(var(C(P), k), σ[k])
                    else
                        remove b from D(y)
                        if D(y) = ∅ then return false
                        add (y, b) to deletionSet
    return true
```

Algorithm 3.2 – fonction GeneralFilteringAlgorithm

```
seekInferableSupport(y : variable, b : value) : tuple
while S_C(y, b) ≠ ∅ do
    σ ← first(S_C(y, b))
    if σ is valid then return σ    /* σ is a support */
    else remove σ from S_C(y, b)
return nil
```

Algorithm 3.3 – fonction seekInferableSupport

trouvé. La liste des tuples contenant (x, a) et supportant une valeur est la liste $S_C(x, a)$; et les valeurs supportées par un tuples τ est donné par $S(\tau)$.

Afin de déterminer les valeurs ayant perdu un support, l'algorithme 3.2 énumère en ligne 1 tous les tuples contenus dans la liste S_C et il énumère en ligne 2 toutes les valeurs supportées par un tuple. Ensuite, l'algorithme essaie de trouver un nouveau support pour ces valeurs soit en "inférant" de nouveaux supports (ligne 3) ou en appelant explicitement la fonction existSolution (ligne 4).

Voici un exemple de fonctionnement de l'algorithme :
Considerons $X = \{x_1, x_2, x_3\}$ et $\forall x \in X D(x) = \{a, b\}$;
avec $T(C(P)) = \{(a, a, a), (a, b, b), (b, b, a), (b, b, b)\}$ (i.e l'ensemble des solutions possibles de P).
Premièrement, un support pour (x_1, a) est recherché : (a, a, a) est calculé et (a, a, a) est ajouté à $S_C(x_2, a)$ et $S_C(x_3, a)$, (x_1, a) dans (a, a, a) est ajouté à $S((a, a, a))$.
Deuxièment, un support pour (x_2, a) est recherché : (a, a, a) appartient à $S_C(x_2, a)$ et il est valide, donc c'est un support, et il n'y a pas besoin de calculer une autre solution.
Ensuite, un support est recherché pour toutes les autres valeurs.
Supposons, maintenant, que la valeur a soit supprimée de x_2, alors tous les tuples de $S_C(x_2, a)$ ne sont plus valides : (a, a, a) par exemple. La validité des valeurs supportées par ce tuple doit être reconsidérée, c'est-à-dire les valeurs appartenant à $S((a, a, a))$, donc un nouveau support pour (x_1, a) doit être recherché et ainsi de suite...

Un programme ayant pour but de réaliser la consistance d'arc pour $C(P)$ doit

créer et initialiser les structures de données (listes S_C et listes S), et appeler la fonction GeneralFilteringAlgorithm($C(P), x, a, deletionSet$) (voir l'algorithme 3.2) chaque fois qu'une valeur a est supprimée du domaine d'une variable x impliquée dans $C(P)$, afin de propager les conséquences de cette suppression. L'ensemble *deletionSet* est mis-à-jour pour contenir les valeurs supprimées qui n'ont pas encore été propagées. Enfin, les listes S_C et S doivent être initialisées de la façon suivante :

- $S_C(x, a)$ contient tous les tuples τ, avec $\tau[x] = a$, qui sont des supports courant pour des valeurs.
- $S(\tau)$ contient toutes les valeurs pour lesquelles τ est le support courant.

La fonction seekInferableSupport de GeneralFilteringAlgorithm recherche un support pour (y, b) dans la liste des tuples supportés par (y, b), dans le but de garantir que l'on ne testera jamais si un tuple est un support pour une valeur si on a déjà fait ce test auparavant. L'idée est d'exploiter la propriété : "Si (y, b) appartient à un tuple supportant une valeur alors ce tuple supporte aussi (y, b)". Les éléments de $S_C(y, b)$ sont donc de bons candidats pour être de nouveaux supports pour (y, b). L'algorithme 3.3 est une implémentation possible de cette fonction.

La complexité de GeneralFilteringAlgorithm est donnée par la table suivante :

	test de consistance		consistance d'arc	
	meilleure	pire	meilleure	pire
A partir du début	$\Omega(P)$	$O(P)$	$nd \times \Omega(P)$	$nd \times O(P)$
Après k modifications	$\Omega(1)$	$k \times O(P)$	$k \times \Omega(1)$	$knd \times O(P)$

La complexité en espace de cet algorithme dépend du nombre de tuples nécessaires pour supporter toutes les valeurs. Or, une valeur est supportée par un seul tuple et il y a nd valeur, donc la complexité en espace est en $O(n^2d)$, où d est la taille du plus grand domaine et n est le nombre de variables impliquées dans la contrainte.

Discussion

L'algorithme 3.2 peut être amélioré si la recherche d'une solution de P est faite selon un ordre préétabli des tuples. Dans ce cas, un algorithme plus complexe est utilisé. On notera alors, que cela revient à traverser un espace de recherche à partir de plusieurs points d'entrée sans jamais traverser deux fois la même partie de cet espace.

De plus, il est possible d'utiliser le solver en lui-même pour calculer une solution de P. Tous ces algorithmes sont détaillés dans [Bessière and Régin, 1997] et [Bessière and Régin, 1999]. Ces articles détaillent également comment l'algorithme 3.2 peut être adapté aux contraintes données par la liste de leurs tuples (dans ce cas la résolution de P revient à recherche un tuple valide dans cette liste) ou par la liste des combinaisons de valeurs interdites pour la contrainte (i.e. la liste complémentaire de la liste précédente).

3.1.3 Perspectives

On peut envisager au moins trois voies de recherche pour améliorer les algorithmes génériques.

Tout d'abord il peut être intéressant de rechercher si l'on peut identifier des structures à l'intérieur de l'ensemble des combinaisons de valeurs satisfaisant la contrainte. Cela permettrait soit de compresser l'information, soit de définir de nouveaux algorithmes exploitant ces structures. Cette dernière approche a récemment été envisagée [Kan Cheng et al., 2003].

Une autre possibilité consiste à gérer des tables de combinaisons de très grandes

tailles, afin de pouvoir utiliser des contraintes dont la définition pose des problèmes à l'heure actuelle. Par exemple, on peut imaginer d'utiliser des techniques connues de compression de données.

Enfin, l'approche calculant l'ensemble des combinaisons autorisées par une contrainte peut s'avérer payante. Cela relance donc l'intérêt de certains problèmes comme celui de la recherche de toutes les solutions d'une contrainte.

3.2 Intégration d'algorithmes de Recherche Opérationnelle en PPC

Dans cette section nous présentons comment certains algorithmes bien connus de RO et de théorie des graphes ont été intégrés en PPC.

Les définitions de la théorie des graphes, de la théorie des couplages et des flots proviennent principalement des livres [Berge, 1970, Lawler, 1976, Tarjan, 1983, Ahuja et al., 1993]. Avant de commencer cette section nous rappelons quelques notions de base de la théorie des graphes.

Un **graphe orienté** ou **digraphe** (abréviation de directed graph en anglais) $G = (X, U)$ est formé par un **ensemble de nœuds** X et un **ensemble d'arcs** U, où un arc est une paire ordonnée de nœuds distincts. Nous noterons $X(G)$ l'ensemble des nœuds du graphe G et $U(G)$ l'ensemble de ses arcs.

Il n'y a pas deux espèces de graphes. Tout graphe est orienté, mais pour des raisons conceptuelles, il est parfois peu commode de le considérer avec son orientation si le problème posé est de nature non orienté. Lorsque l'orientation n'est pas précisée on parle alors d'arête plutôt que d'arcs. Chaque fois qu'on appliquera un concept orienté à un graphe défini à partir d'arêtes, ce concept devra être appliqué en fait au graphe orienté qui lui correspond en orientant dans les deux sens chaque arête. De même chaque fois qu'on appliquera un concept non orienté à un graphe, ce concept devra être appliqué en omettant les orientations.

Un **chemin** (chaîne dans le cas non orienté) d'un nœud v_1 à un nœud v_k dans G est une liste de nœuds $[v_1, ..., v_k]$ telle que (v_i, v_{i+1}) soit un arc (une arête dans le cas non orienté) pour $i \in [1..k-1]$. Le chemin **contient** les nœuds v_i pour $i \in [1..k]$ et les arcs (v_i, v_{i+1}) pour $i \in [1..k-1]$. Le chemin est **élémentaire** si tous ces nœuds sont distincts. Le chemin est un **circuit** si $k > 1$ et $v_1 = v_k$.

3.2.1 Couplage biparti et Contrainte "Alldiff"

Présentation

La contrainte alldiff impose que les valeurs prises par un ensemble de variables soient deux à deux différentes.

Définition 2 *Une* **contrainte alldiff** *est une contrainte C telle que*
$T(C) = \{\tau$ *tel que* τ *est un tuple de* $X(C)$ *et* $\forall a_i \in D(X(C)) : \#(a_i, \tau) \leq 1\}$
où $\#(a_i, \tau)$ *désigne le nombre d'occurrences* a_i *dans* τ*.*

Cette contrainte est utilisée dans un grand nombre de problèmes réels, comme l'allocation de ressource ou les problèmes d'emploi du temps. Dès lors que deux choses ne peuvent pas se trouver au même endroit au même moment on introduira une contrainte alldiff.

Résultats

J'ai montré que l'on peut définir un algorithme de filtrage réalisant la consistance d'arc pour une contrainte alldiff en utilisant la théorie des couplages [Régin, 1994]. Aussi, il est nécessaire de rappeler les bases de cette théorie.

Un ensemble d'arêtes d'un graphe G tel qu'il n'existe pas deux arêtes ayant un sommet en commun est appelé **couplage**. Un couplage de cardinalité maximum est appelé un couplage maximum. Un couplage M couvre un ensemble X de nœuds de G si tout nœud de X est l'extrémité d'une arête de M.

On remarquera qu'un couplage qui couvre X dans le graphe biparti $G = (X, Y, E)$ est un couplage maximum.

La relation entre un couplage et une contrainte alldiff se fait au travers du graphe des valeurs de la contrainte.

Définition 3 ([Laurière, 1978]) *Soit C une contrainte, le graphe biparti $GV(C) = (X(C), D(X(C)), E)$ où $(x, a) \in E$ si et seulement si $a \in D(x)$ est appelé le* **graphe des valeurs** *of C.*

On a alors le théorème suivant :

Théorème 1 ([Régin, 1994]) *Etant donné une contrainte alldiff C. C est consistante si et seulement si il existe un couplage couvrant X dans $GV(C)$.*

L'utilisation de la théorie des couplages est particulièrement intéressante car on dispose d'algorithmes efficaces calculant un couplage maximum dans un graphe biparti. Par exemple [Hopcroft and Karp, 1973] ont proposé un algorithme dont la complexité est en $O(\sqrt{|X|}m)$, où m est le nombre d'arêtes du graphe.

On peut ensuite décrire simplement un algorithme de filtrage associé à une contrainte alldiff qui va réaliser la consistance d'arc pour cette contrainte, c'est-à-dire caractériser et identifier facilement les valeurs qui ne sont pas consistantes avec la contrainte, autrement dit les valeurs a d'une variable x pour lesquelles il n'existe pas de couplage contenant l'arête (x, a) et couvrant X. Pour parvenir à ce résultat, nous devons rappeler quelques définitions.

Soit M un couplage. Une arête de M est dite **couplée**, alors qu'une arête qui n'appartient pas à M est dite **libre**. Un nœud est **couplé** s'il est l'extrémité d'une arête de M, sinon il est **libre**. Une **chaîne alternée**, respectivement un **cycle alterné**, est une chaîne élémentaire, respectivement un cycle élémentaire, dont les arêtes sont alternativement couplées et libres. La **longueur** d'une chaîne ou d'un cycle alterné est le nombre d'arêtes qu'il contient. Une arête appartenant à tous les couplages maximum est dite **vitale**.

Propriété 2 ([Berge, 1970]) *Une arête appartient à des couplages maximum mais non à tous si et seulement si pour un couplage maximum arbitraire M, cette arête appartient soit à une chaîne alternée paire qui commence à un nœud libre, soit à un cycle alterné pair.*

A partir de cette propriété on peut facilement établir une propriété sur la consistance d'une valeur avec une contrainte alldiff.

Proposition 2 *Soient C une contrainte alldiff, a une valeur d'une variable x impliquée dans C et M un couplage couvrant $X(C)$ dans $GV(C)$. (x, a) est consistante avec C si et seulement si l'une des conditions suivantes est vraie :*

- *l'arête (x, a) appartient à M,*
- *l'arête (x, a) appartient à une chaîne alternée paire de $GV(C)$ qui commence à un nœud libre,*
- *l'arête (x, a) appartient à un cycle alterné pair de $GV(C)$.*

Or, si l'on oriente les arêtes de $GV(C)$ dans le sens des valeurs vers les variables si elles appartiennent à M et dans le sens inverse sinon, alors on peut facilement calculer en tenant compte de l'orientation celles qui appartiennent à une chaîne alternée paire et celles qui appartiennent à un cycle alterné pair. Pour réaliser cela il

suffit d'employer une recherche en profondeur d'abord à partir des nœuds libres afin de déterminer les chaînes alternées paires et de calculer les composantes fortement connexes du graphe orienté. Ces deux opérations se faisant en temps linéaire ($O(m)$, m étant le nombre d'arêtes du graphe) on obtient donc un algorithme efficace pour caractériser chacune des arêtes de la proposition précédente. En conséquence, on peut supprimer les valeurs non consistantes avec la contrainte en temps linéaire ([Régin, 1994]).

L'avantage de cette méthode est qu'elle est incrémentale. Si certaines valeurs sont supprimées par d'autres contraintes alors on peut calculer un nouveau couplage en utilisant le précédent.

Travaux connexes

A peu près tous les solvers existants intègrent l'algorithme qui vient d'être presénté. Cet algorithme est l'un des premiers qui a propose d'utiliser des techniques de Recherche Opérationnelle afin de calculer la consistance d'arc d'une contrainte. Le succès de cet algorithme a donné une forte impulsion à un thème naissant en PPC : les contraintes globales. Depuis, de nouvelles contraintes globales encapsulant des algorithmes de RO ou de théorie des graphes sont publiées chaque année dans les conférences de PPC ou d'IA sur ce thème. De nombreuses études comparant les différents niveaux de filtrage ont aussi été proposées.

Plusieurs travaux ont été ensuite entrepris pour la contrainte alldiff en considérant le cas où les domaines des variables sont des intervalles d'entiers. Il est cependant important de remarquer que même dans ce cas il est possible de créer un nombre quadratique de "trous" dans les domaines. Par exemple, considérons une contrainte alldiff définie sur $X = \{x_1, ..., x_n\}$ avec les domaines : $\forall i \in [1, \frac{n}{2}]$, si i est impair alors $D(x_i) = [2i - 1, 2i]$ sinon $D(x_i) = D(x_{i-1})$; et $\forall i \in [\frac{n}{2} + 1, n]$ $D(x_i) = [1, n]$. Ainsi, pour $n = 12$ on a : $D(x_1) = D(x_2) = [1, 2]$, $D(x_3) = D(x_4) = [5, 6]$, $D(x_5) = D(x_6) = [9, 10]$, $D(x_7) = D(x_8) = D(x_9) = D(x_10) = D(x_11) = D(x_12) = [1, 12]$. Alors, si l'on réalise la consistance d'arc les intervalles correspondants aux variables entre x_1 et $x_{\frac{n}{2}}$ seront supprimés des domaines des variables de $x_{\frac{n}{2}+1}$ à x_n. C'est-à-dire, $2 \times \frac{n}{2}$ valeurs seront effectivement supprimées du domaine de $(n - (\frac{n}{2} + 1))$ variables. Aussi, $O(n^2)$ valeurs sont éliminées. Comme m est borné par n^2, l'algorithme de filtrage que nous avons présenté peut être considéré comme un algorithme optimal dans le pire des cas.

[Leconte, 1996] a proposé de considérer que les domaines des variables sont des intervalles et d'effectuer toutes les suppressions possibles. Cet algorithme est basé sur les principes de l'edge-finder et sa complexité est en $O(n^2 d)$. Ensuite, [Bleuzen-Guernalec and Colmerauer, 1997] ont proposé de ne mettre à jour que les bornes. Leur algorithme a été amélioré par [Puget, 1998] pour atteindre $O(n \log(n))$. Puis, [Melhorn and Thiel, 2000] ont donné un algorithme de filtrage par consistance de bornes avec une complexité qui est asymptotiquement la même que celle du tri d'un ensemble d'intervalles. Si les intervalles sont des entiers entre 0 et $O(n^k)$ pour une constante k alors l'algorithme est linéaire. En fait, cet algorithme est le même que celui pour le cas général qui exploite de façon efficace les spécificités du cas particulier considéré. Enfin, [Lopez-Ortiz et al., 2003] ont exhibé un algorithme original et simple de même complexité. [Stergiou and Walsh, 1999] ont fait une comparaison des différents algorithmes disponibles et montré leur intérêt en pratique.

Notons également que [Bleuzen-Guernalec and Colmerauer, 1997] ont étudié des contraintes proches comme la contrainte de permutation (il y autant de valeurs que de variables) et une contrainte de tri. Dans le cas de la permutation l'algorithme donné par [Melhorn and Thiel, 2000] est linéaire.

Perspective

Il pourrait être intéressant d'essayer de combiner une contrainte alldiff avec d'autres contraintes. Par exemple, avec des contraintes binaires ou ternaires simples. Même s'il n'est pas nécessairement question de donner des algorithmes de filtrage réalisant la consistance d'arc. Il serait au moins intéressant d'exhiber certaines limites à ce type de combinaisons. On peut sans prendre trop de risque considérer que des combinaisons très particulières auraient certainement un intérêt en pratique. Il reste à déterminer lesquelles.

3.2.2 Couplage et Contrainte "Symmetric alldiff"

Présentation

La contrainte symmetric alldiff impose le groupement par paires d'entités. C'est un cas particulier de la contrainte alldiff pour lequel les variables et les valeurs sont définis à partir du même ensemble S. Chaque variable représente un élément e de S et ses valeurs correspondent aux élément de S qui sont compatibles avec e. Cette contrainte requiert que toutes les valeurs prises par les variables soient deux à deux différentes (comme pour la contrainte alldiff) et que si une variable représentant l'élément i est affecté à une valeur j, alors la variable représentant l'élément j doit être affectée à la valeur i. Formellement on a :

Définition 4 *Soient X un ensemble de variables et σ une bijection de $X \cup D(X)$ dans $X \cup D(X)$ telle que*
$\forall x \in X : \sigma(x) \in D(X)$; $\forall a \in D(X) : \sigma(a) \in X$ *and* $\sigma(x) = a \Leftrightarrow x = \sigma(a)$.
Une **contrainte symmetric alldiff** *définie sur X est une contrainte C associée à σ telle que :*
$T(C) = \{ \tau$, *où* τ *est un tuple de* X
et $\forall a \in D(X) : \#(a, \tau) = 1$
et $a = \tau[index(C, x)] \Leftrightarrow \sigma(x) = \tau[index(C, \sigma(a))]$

Cette contrainte est utile lorsqu'il faut grouper des entités par paires. On la retrouve donc dans des problèmes comme les problèmes d'emploi du temps, d'affectation d'équipages ou de calendrier sportifs.

Résultats

J'ai proposé d'étudier cette contrainte.

Le test de la consistance de cette contrainte est équivalent à la recherche d'un couplage couvrant tous les nœuds dans le graphe non-biparti $G = (S, E)$, où S est l'ensemble des éléments à grouper par paires et deux éléments sont reliés entre eux si et seulement si il est possible de former une paire contenant ces deux éléments [Régin, 1999b]. Ce problème peut être résolu en $O(\sqrt{n}m)$ en utilisant l'algorithme complexe de [Micali and Vazirani, 1980]. Le problème de la recherche d'un couplage dans un graphe non-biparti est parfois appelé couplage symmétrique, car il peut être vu comme la recherche d'un couplage dans le graphe biparti $G = (S, S, E)$ qui tient compte de la contrainte imposant que si l'arête (i, j) appartient au couplage alors l'arête (j, i) doit également appartenir au couplage. C'est pourquoi, cette contrainte porte le nom de symmetric alldiff.

Ce qui est remarquable c'est que la propriété de Berge reste valide dans le cas d'un graphe non biparti (cf propriété 2). Donc, on a la proposition suivante qui est très proche de celle du alldiff (dans ce cas précis il ne peut pas exister de nœuds libres) :

Proposition 3 *Soient C une contrainte symmetric alldiff, a une valeur d'une variable x impliquée dans C et M un couplage couvrant S dans $G = (S, E)$. (x, a) est consistante avec C si et seulement si l'une des conditions suivantes est vraie :*

- *l'arête (x, a) appartient à M,*
- *l'arête (x, a) appartient à un cycle alterné pair de $GV(C)$.*

Malheureusement, le principe de l'orientation des arêtes utilisé pour la contrainte alldiff n'est plus valide pour les graphes non biparti. Néanmoins, il est possible d'identifier l'ensemble des valeurs non consistantes avec la contrainte en $O(nm)$. Cette complexité est nettement moins bonne que celle du alldiff, mais elle peut s'avérer acceptable en pratique.

Dans l'article [Régin, 1999b], un autre algorithme de filtrage est proposé. Cet algorithme ne réalise plus la consistance d'arc, mais il a une complexité qui s'amortit pour les suppressions ($O(m)$ par suppression). le filtrage réalisé par cet algorithme ne peut, malheureusement, pas être caractérisé. Enfin, cet article propose aussi de traiter cette contrainte comme la conjonction d'une contrainte alldiff et de contrainte additionnelle portant sur la parité des composantes 2-connexes du graphe.

Travaux connexes

Une comparaison entre les différents algorithmes de filtrage possibles pour cette contrainte a été faite par [Henz et al., 2003]. Cette comparaison montre qu'il existe pour chaque algorithme de filtrage un type de problème pour lequel l'algorithme considéré est le plus efficace.

Perspectives

Il serait bien sûr intéressant d'obtenir pour la contrainte symmetric alldiff un résultat similaire à celui obtenu pour la contrainte alldiff, c'est-à-dire que la complexité de l'algorithme de filtrage réalisant la consistance d'arc soit du même ordre de grandeur que celle du test de la consistance.

3.2.3 Flot et Contrainte globale de cardinalité

Présentation

Une *contrainte globale de cardinalité* (GCC) contraint le nombre d'affectation de variables par valeur. Cette contrainte est certainement l'une des plus utiles en pratique. On remarquera qu'une contrainte alldiff correspond à une GCC pour laquelle chaque valeur doit être prise au plus une fois. Formellement on a :

Définition 5 *Une* **contrainte globale de cardinalité** *est une contrainte C associée à un ensemble V de valeurs, avec $D(X(C))) \subseteq V$ pour laquelle chaque valeur $a_i \in V$ est associée à deux entiers positifs l_i et u_i avec $l_i \leq u_i$ et telle que*
$T(C) = \{ \tau$ *où* τ *est un tuple de* $X(C)$
et $\forall a_i \in V : l_i \leq \#(a_i, \tau) \leq u_i\}$
Elle est notée $gcc(X, V, l, u)$.

Cette contrainte est présente dans tous les problèmes d'emploi du temps ou d'ordonnancement de voitures sur une chaîne de montage.

Résultats

J'ai proposé un algorithme réalisant la consistance d'arc pour cette contrainte [Régin, 1996], c'est-à-dire supprimant toutes les valeurs qui n'appartiennent pas à une solution de la contrainte. Cet algorithme est basé sur l'utilisation de la théorie des flots. Aussi, il est nécessaire d'en rappeler les principes.

Soit G un graphe pour lequel chaque arc (i, j) est associé à deux entiers l_{ij} et u_{ij}, respectivement appelés la **borne inférieure de capacité** et la **borne supérieure de capacité** d'un arc.

Un **flot** dans G est une fonction f qui satisfait les deux conditions suivantes :

• Pour tout arc (i, j), f_{ij} représente la quantité de commodité qui peut traverser l'arc. Un flot n'est permis que dans le sens indiqué par l'arc, i.e., de i vers j. Pour simplifier l'exposé nous supposerons que $f_{ij} = 0$ si $(i, j) \notin U(G)$.

• Une **loi de conservation** est observé en chaque nœud : $\forall j \in X(G) : \sum_i f_{ij} = \sum_k f_{jk}$.

Nous considérerons dans cette section deux problèmes de la théorie des flots :

• **le problème du flot compatible** : Existe-t'il un flot dans G qui satisfait les **contraintes de capacité** ? C'est-à-dire, trouver un flot f tel que $\forall (i, j) \in U(G)$ $l_{ij} \leq f_{ij} \leq u_{ij}$.

• **le problème du flot maximum traversant un arc** (i, j) : Trouver un flot compatible dans G pour lequel la valeur de f_{ij} est maximum.

Sans perte de généralité (voir p.45 et p.297 in [Ahuja et al., 1993]), et pour pouvoir éviter des problèmes de notations, nous considérerons que :

• si (i, j) est un arc de G alors (j, i) n'est pas un arc de G.

• toutes les bornes de capacités sont des entiers positifs ou nuls.

En fait si toutes les bornes sont des entiers et s'il existe un flot compatible, alors il en existe un qui a une valeur entière sur chaque arc de G (voir [Lawler, 1976] p113).

On peut alors montrer que tester la consistance d'une GCC est équivalent à rechercher s'il existe un flot compatible dans un graphe particulier, appelé le réseau de valeurs de la contrainte [Régin, 1996] :

Définition 6 *Soit $C = gcc(X, V, l, u)$ une GCC ; le* **réseau de valeurs** *de C est le graphe biparti orienté $N(C)$ munit de bornes inférieures et supérieures de capacité sur chaque arc. L'ensemble des nœuds de $N(C)$ est formé par l'ensemble X, l'ensemble V et deux nœuds supplémentaires s et t. Les arcs de $N(C)$ sont définis de la façon suivante :*

• *il existe un arc entre une valeur a de V et une variable x de X si et seulement si $a \in D(x)$. Pour chacun de ces arcs (a, x) on a $l_{ax} = 0$ et $u_{ax} = 1$.*

• *il existe un arc entre s et toutes les valeurs a_i de V. Pour chaque arc (s, a_i) on a $l_{sa_i} = l_i$ et $u_{sa_i} = u_i$.*

• *il existe un arc entre toutes les variables x de X et t. Pour chaque arc (x, t) on a $l_{xt} = 1$, $u_{xt} = 1$.*

• *il existe un arc (t, s) avec $l_{ts} = u_{ts} = |X(C)|$.*

On remarquera que le réseau de valeurs est orienté dans le sens des valeurs vers les variables.

Proposition 4 ([Régin, 1996]) *Soient C une GCC et $N(C)$ le réseau de valeurs de C. Les deux propriétés suivantes sont équivalentes*

• *C est consistante ;*

• *il existe un flot compatible dans $N(C)$.*

Afin de proposer un algorithme réalisant la consistance d'arc, nous avons besoin de rappeler une autre notion de la théorie des flots :

Définition 7 *Le* **graphe résiduel** *d'un flot f, noté $R(f)$, est le graphe orienté ayant le même ensemble de nœuds que G. L'ensemble des arcs de $R(f)$ est défini*

comme suit :
$\forall (i,j) \in U(G)$ *:*

• $f_{ij} < u_{ij} \Leftrightarrow (i,j) \in U(R(f))$ *et sa borne supérieure de capacité est* $r_{ij} = u_{ij} - f_{ij}$.

• $f_{ij} > l_{ij} \Leftrightarrow (j,i) \in U(R(f))$ *et sa borne supérieure de capacité est* $r_{ji} = f_{ij} - l_{ij}$.

Toutes les bornes inférieures de capacité sont nulles.

On peut alors établir la proposition suivante :

Proposition 5 ([Régin, 1996]) *Soient C une GCC consistante et f un flot compatible de $N(C)$. Une valeur a d'une variable x n'est pas consistante avec C si et seulement si $f_{ax} = 0$ et a et x appartiennent à des composantes fortement connexes différentes de $R(f)$.*

L'avantage de cette proposition est que toutes les valeurs non consistantes avec la contrainte peuvent être déterminées en identifiant une et une seule fois les composantes fortement connexes de $R(f)$.

La recherche d'un flot compatible peut être calculé en $O(nm)$, c'est donc la complexité du test de consistance d'une GCC. La recherche des composantes fortement connexes peut être effectuée en $O(m + n + d)$ [Tarjan, 1983], aussi on obtient la même complexité pour éliminer toutes les valeurs non consistantes avec une GCC.

Enfin, notons que les algorithmes de flots sont incrémentaux.

Travaux connexes

Deux algorithmes réalisant la borne consistance d'une GCC ont été développé en parallèle [Quimper et al., 2003] et [Katriel and Thiel, 2003]. Le premier algorithme est original alors que le second est une adaptation de l'algorithme que nous avons présenté au cas particulier qui est considéré (toutes les variables ont des domaines qui sont des intervalles et on ne cherche qu'à mettre à jour leurs bornes).

Perspectives

La contrainte globale de cardinalité implique un ensemble d'intervalles. On peut imaginer que ces intervalles soient obtenus à partir des bornes de variables. On peut alors s'intéresser aux filtrages des domaines de ces variables. Cela a été envisagé par [Katriel and Thiel, 2003], mais pas dans le cas général. Un travail important reste donc à réaliser dans ce domaine.

On peut aussi considérer une version plus générale de la problématique sous-jacente à cette contrainte. Au lieu de ne considérer que des arcs de type $(0,1)$, autrement dit acceptant au plus une unité de flot, on pourrait étudier une forme plus générale du problème dans laquelle ces arcs peuvent prendre n'importe quelle valeur. Il s'agirait de déterminer pour chaque arc quelles sont les quantités minimales et maximales de flots qui peuvent passer par cet arc. L'idée étant, bien sûr, d'essayer d'obtenir un algorithme global, c'est-à-dire qui évite de considérer les arcs de façon indépendante. On s'approche alors du domaine de la "sensivity analysis" en anglais.

3.2.4 Flot à coût minimum et Contrainte globale de cardinalité avec coûts

Présentation

Une contrainte globale de cardinalité avec coûts (costGCC) est la conjonction d'une contrainte globale de cardinalité et d'une contrainte de somme portant les coûts des affectations.

Définition 8 *Une* **fonction de coût sur un ensemble de variable** X *est une fonction qui associe à chaque valeur* (x, a), $x \in X$ *et* $a \in D(x)$ *un entier noté* $cost(x, a)$.

Définition 9 *Une* **contrainte globale de cardinalité avec coûts** *est une contrainte* C *associée à un ensemble de valeurs* V, `cost` *une fonction de coût sur* $X(C)$, *un entier* H *et pour laquelle chaque valeur* $a_i \in V$ *est associé avec deux entiers positifs* l_i *et* u_i, *et telle que*
$T(C) = \{\ \tau$ *où* τ *est un tuple de* $X(C)$
et $\forall a_i \in V : l_i \leq \#(a_i, \tau) \leq u_i$
et $\Sigma_{i=1}^{|X(C)|} cost(var(C, i), \tau[i]) \leq H\ \}$
Elle est notée $costgcc(X, V, l, u, cost, H)$.

Cette contrainte est utilisée pour modéliser des préférences entre affectations dans les problèmes d'affectation de ressources. On remarquera qu'aucune supposition n'est faite sur le signe des coûts.

La prise en compte de coûts par une contrainte est particulièrement important notamment pour résoudre des problèmes d'optimisations, parce que cela peut considérablement améliorer la propagation dûe à une modification des variables de coût. Autrement dit, le domaine des variables peut être réduit lorsque la variable objectif est modifiée.

Résultats

J'ai proposé un algorithme de filtrage réalisant la consistance d'arc pour cette contrainte. Cet algorithme est basé sur la recherche de flots à coût minimum. Afin de présenter les idées de cet algorithme, nous rappelons quelques notions sur les flots à coût minimum.

Soit G un graphe dont les arcs sont munis de contraintes de capacités (voir section précédente) et d'un entier qui représente le coût de traversée pour une unité de flot. Le coût d'un flot est défini par $cost(f) = \sum_{(i,j) \in U(G)} f_{ij} c_{ij}$.

Le problème du flot compatible à coût minimum est le suivant : S'il existe un flot compatible dans G, trouver un flot compatible f tel que $cost(f)$ soit minimum.

Le test de consistance d'une contrainte costGCC s'obtient en recherchant s'il existe un flot compatible dont la valeur est strictement inférieure à H dans le réseau de valeurs associé à la contrainte qui est défini de la façon suivante :

Définition 10 ([Régin, 1999a]) *Etant donné* $C = costgcc(X, V, l, u, cost, H)$ *; le* **réseau de valeurs** $N(C)$ *de* C *est le réseau de valeur* $N(C')$ *de la contrainte globale de cardinalité sous-jacente* $C' = gcc(X, V, l, u)$ *de* C, *dans lequel chaque arc est muni d'un coût défini par :*

- $\forall a \in V : c_{sa} = 0$
- $\forall x \in X(C) : c_{xt} = 0$
- $c_{ts} = 0$
- $\forall x \in X(C) : \forall a \in D(x) : c_{ax} = cost(x, a)$.

Proposition 6 ([Régin, 1999a]) *Soient* $C = costgcc(X, V, l, u, cost, H)$ *et* $N(C)$ *le réseau de valeur de* C *; les propriétés suivantes sont équivalentes :*

- C *est consistante ;*
- *il existe un flot compatible dans* $N(C)$ *dont le coût est strictement inférieur à* H.

La recherche d'un tel flot peut se faire en utilisant l'algorithme de recherche de plus courts chemins successifs ("the successive shortest paths algorithm" en anglais). La complexité de cet algorithme est $O(nS(m, n+d, \gamma))$, où $S(m, n+d, \gamma)$ est la complexité de la recherche d'un plus court chemin dans un graphe ayant m arcs, $n+d$ sommets et dont la plus grande valeur du coût est γ. En fait, la complexité de la recherche d'un plus court chemin dans un graphe ayant m arcs et n nœuds dépend de la valeur maximale du coût des arcs ainsi que du signe des coûts. Nous noterons cette complexité $S(m, n, \gamma)$ si tous les coûts sont positifs ou nuls et $S_{neg}(m, n, \gamma)$ sinon.

L'intérêt de l'utilisation de cet algorithme est son aspect incrémental. Si k valeurs sont supprimées des domaines alors on peut tester la consistance à partir du flot précédant avec une complexité en $O(kS(m, n+d, \gamma))$.

L'algorithme réalisant le filtrage par consistance d'arc utilise le graphe résiduel, qui lorsque l'on introduit des coûts sur les arcs se définit de la façon suivante :

Définition 11 *Le* **graphe résiduel** *d'un flot f dans G muni de coûts sur les arcs, noté $R(f)$, est le graphe orienté ayant le même ensemble de nœuds que G et dont l'ensemble des arcs est défini par :*
$\forall (i,j) \in U(G)$:

- *$f_{ij} < u_{ij} \Leftrightarrow (i,j) \in U(R(f))$ avec un coût $rc_{ij} = c_{ij}$ et une borne supérieure de capacité $r_{ij} = u_{ij} - f_{ij}$.*
- *$f_{ij} > l_{ij} \Leftrightarrow (j,i) \in U(R(f))$ avec un coût $rc_{ji} = -c_{ij}$ et une borne supérieure de capacité $r_{ji} = f_{ij} - l_{ij}$.*

Toutes les bornes inférieures de capacité sont nulles.

Nous considérerons par la suite que f^o est un flot compatible à coût minimum dans $N(C)$. En notant $d_G(x, y)$ la distance du plus court chemin de x à y dans le graphe G, on peut alors établir la proposition suivante :

Proposition 7 ([Régin, 1999a]) *Une valeur a d'une variable y n'est pas consistante avec C si et seulement si :*
$f^o_{ay} = 0$ *et* $d_{R(f^o)-\{(y,a)\}}(y, a) > H - cost(f^o) - rc_{ay}$

Cette proposition peut être modifiée en tirant parti du fait que lorsqu'on recherche un chemin de y à a, l'arc (y, a) n'appartient pas à $R(f^0)$ puisque $f^o_{ay} = 0 = l_{ay}$. On a donc $R(f^o) - \{(y, a)\} = R(f^o)$.

Corollaire 1 ([Régin, 1999a]) *Une valeur a d'une variable y n'est pas consistante avec C si et seulement si :*

$$f^o_{ay} = 0 \text{ et } d_{R(f^o)}(y, a) > H - cost(f^o) - rc_{ay}.$$

Aussi, si pour une variable y on calcule les plus courts chemins de y à chaque nœud de $R(f^o)$, alors on pourra déterminer les valeurs de y qui ne sont pas consistantes avec la contrainte. Donc, comme il y a n variables, on peut éliminer toutes les valeurs non consistantes avec la contrainte en $O(nS_{neg}(m, n+d, \gamma))$.

On peut utiliser davantage la structure particulière de $R(f^0)$. Dans une solution chaque variable est affectée à une valeur. Cela signifie que pour chaque variable il y a seulement un et un seul arc entrant dans y dans $R(f^o)$. Tous les plus courts chemin traversant cette variable vont donc utiliser cet arc. Ainsi, si y est affecté à b alors tous les plus courts chemins dans $R(f^o)$ de y à une valeur a utilisent l'arc (y, b) et on a $d_{R(f^o)}(y, a) = rc_y b + d_{R(f^o)}(b, a)$. On peut donc déterminer les valeurs de y qui ne sont pas consistantes avec C en recherchant des plus courts chemins dans $R(f^o)$ à partir de b, la valeur affectée à y.

Corollaire 2 ([Régin, 1999a]) *Soit y une variable telle que $f^o_{by} = 1$. Alors, une valeur a de y n'est pas consistante avec C si et seulement si*
$$f^o_{ay} = 0 \text{ et } d_{R(f^o)}(b, a) > H - cost(f^o) - rc_{ay} - rc_{yb}.$$

L'avantage de cette méthode (i.e. calculer les plus courts chemins à partir des valeurs et non pas des variables) est que l'on peut regrouper des calculs, puisque plusieurs variables peuvent être affectées à la même valeur.

Soit Δ l'ensemble des valeurs b telles que $f^o_{sb} > 0$. Considérons b l'une de ces valeurs , on notera $\delta(b) = \{a \in D(X(C)) \text{ où } a \neq b \text{ et } a \in D(y) \text{ et } f^o_{by} = 1\}$. Alors, la consistance d'arc peut être réalisée en recherchant pour chaque valeur b de Δ les plus courts chemins de b à chaque valeur de $\delta(b)$.

La complexité des algorithmes précédents dépend du signe des coûts, parce que les meilleurs algorithmes de recherche de plus court chemin n'acceptent que des coûts positifs ou nuls. Or, le graphe résiduel contient des coûts négatifs. Cependant, il est possible de modifier ce graphe afin de ne plus avoir de tels coûts.

Soit f le flot courant, et considérons que les plus courtes distances à partir d'un nœud s ont été calculé. On a alors la propriété bien connue des plus courts chemins :
$$\forall i, j \in X(R(f)) : d_{R(f)}(s, j) \leq d_{R(f)}(s, i) + rc_{ij}.$$

Donc $\forall i, j \in X(R(f)) : d_{R(f)}(s, i) + rc_{ij} - d_{R(f)}(s, j) \geq 0$. On peut remplacer chaque coût du graphe résiduel par $\overline{c}^s_{ij} = d_{R(f)}(s, i) + rc_{ij} - d_{R(f)}(s, j)$. Ces coûts sont souvent appelés **coûts réduits**. On notera $\overline{R}^s(f)$ ce graphe résiduel modifié. La relation entre ce graphe et le graphe résiduel originel est donné par la propriété suivante :

Propriété 3 *Soient f un flot et $\overline{R}^s(f)$ le graphe résiduel modifié de f dans lequel les coûts sont définis par $\overline{c}^s_{ij} = d_{R(f)}(s, i) + rc_{ij} - d_{R(f)}(s, j)$.*
Alors, $\forall (i, j) \in U(\overline{R}^s(f))\ \overline{c}^s_{ij} \geq 0$,
et $d_{R(f)}(u, v) = d_{\overline{R}^s(f)}(u, v) - d_{R(f)}(s, u) + d_{R(f)}(s, v)$.

Tous les coûts de $\overline{R}^s(f)$ sont positifs ou nuls, donc les meilleur algorithmes pour calculer les plus courts chemins peuvent être utilisés.

On obtient alors le corollaire suivant :

Corollaire 3 ([Régin, 1999a]) *Soit y une variable telle que $f^o_{by} = 1$. Alors, une valeur a de y n'est pas consistante avec C si et seulement si*
$$f^o_{ay} = 0 \text{ et } d_{\overline{R}^t(f^o)}(b, a) > H - cost(f^o) - \overline{c}^t_{ay} - \overline{c}^t_{yb}.$$

Comme $|\Delta| \leq min(n, d)$, on obtient alors :

Propriété 4 ([Régin, 1999a]) *Soient C une contrainte costGCC consistante, f^o un flot à coût minimum dans $N(C)$. La consistance d'arc de C peut être réalisée en $O(|\Delta| S(m, n + d, \gamma))$.*

Travaux connexes

[Caseau and Laburthe, 1997] ont utilisé une contrainte alldiff avec coûts, mais seule la consistance de cette contrainte a été testé, aucun algorithme de filtrage spécifique n'a été proposé. Le premier algorithme de ce type a été donné par [Focacci et al., 1999a] et [Focacci et al., 1999b]. Cet algorithme ne réalise pas la consistance d'arc, il propose une réduction de domaine basée sur l'utilisation des coûts réduits. Dans [Régin, 1999a] et [Régin, 2002] nous avons amélioré cet algorithme. Enfin, notons l'algorithme précédemment décrit est le premier qui réalise la consistance d'arc.

Perspectives

Une contrainte globale de cardinalité avec coûts est la combinaison d'une gcc et d'une contrainte de somme bornée. Le coût est en fait un représentant d'un critère quelconque qui doit être additif. On peut alors se demander s'il ne serait pas possible d'essayer de considérer plusieurs critères en même temps, c'est-à-dire plusieurs contraintes de somme bornée, comme on peut le faire pour les problèmes de plus courts chemins sous contraintes.

3.3 Contraintes dérivées

Les contraintes intégrant des algorithmes de flots sont très puissantes puisqu'elles sont plus générales que de nombreuses contraintes. Cette section a pour but de montrer comment certaines contraintes s'expriment sous la forme d'une des contraintes que nous avons présentées dans la section précédente.

3.3.1 Alldiff avec coûts

Présentation

Cette contrainte est la conjonction d'une contrainte alldiff et d'une contrainte de somme portant sur les coûts des affectations.

Résultats

Cette contrainte s'exprime immédiatement sous la forme d'une contrainte globale de cardinalité avec coûts, pour laquelle chaque valeur doit être prise au plus une fois. Le filtrage réalisant la consistance d'arc pour la contrainte costGCC réalise évidemment aussi la consistance d'arc pour cette contrainte.

3.3.2 Contrainte de plus court chemin

Présentation

Soient un graphe G, dont les arcs sont munis de coûts, H un entier et s et t deux nœuds particuliers de G. Cette contrainte impose l'existence d'un chemin de s à t dans G dont le coût est strictement inférieur à H.

On notera que la recherche d'un chemin élémentaire empruntant un arc ou un nœud donné est un problème NP-Complet. Il n'est donc pas possible, à l'heure actuelle, de donner un algorithme de filtrage par consistance d'arc polynomial pour cette contrainte. L'aspect élémentaire des chemins n'est donc pas pris en compte.

Résultats

Le problème du flot à coût minimum est plus général que celui du plus court chemin. En fait, ce dernier problème peut être exprimé sous la forme d'un problème de flot à coût minimum. Il s'agit d'envoyer une unité de flot d'un nœud s à un nœud t, dans un graphe dont toutes les bornes supérieures de capacité des arcs valent 1.

On peut représenter G à l'aide d'un ensemble de variables X comme suit :

- à chaque nœud de G correspond une variable
- le domaine de chaque variables est constitué des nœuds voisins de G auquel on ajoute le nœud correspondant à la variable (ce qui permet de dire que le nœud n'appartient pas au chemin), sauf pour la variable correspondant au nœuds t, dont le domaine ne contient que le noeuds s.

Cette contrainte se modélise alors par une contrainte costGCC définie sur X, les bornes supérieures de capacité sont toutes égales à 1 et les bornes inférieures de capacités valent 0 pour tous les nœuds sauf pour s et t pour lesquels elles valent 1.

Le filtrage réalisant la consistance d'arc pour la costGCC peut donc être utilisé comme filtrage pour cette contrainte. Il réalise la consistance d'arc pour cette contrainte si l'élémentarité des chemins n'est pas considérée.

3.3.3 Somme et produit scalaire de variables toutes différentes

Présentation

Pour un ensemble de variables X, cette contrainte est la conjonction d'une contrainte $\sum_{x_i \in X} x_i \leq H$ et alldiff(X), ou plus généralement la conjonction de $\sum_{x_i \in X} \alpha_i x_i \leq H$ et de alldiff(X).

Cette contrainte est utile, par exemple, pour résoudre le problème de la règle de golomb (golomb ruler).

Formellement on a :

Définition 12 *Une contrainte* **produit scalaire de variables toutes différentes** *est une contrainte C associée avec α un ensemble de coefficients, un pour chaque variable, et un entier H tel que :*

$T(C) = \{\ \tau$ *où* τ *est un tuple de* $X(C)$
et $\forall a_i \in D(X(C)) : \#(a_i, \tau) \leq 1$
et $\Sigma_{i=1}^{|X(C)|} \alpha_i \tau[i] \leq H\ \}$

Résultats

Cette contrainte peut se modéliser sous la forme d'une contrainte costGCC en définissant les bornes de capacité et les coûts de la façon suivante [Régin, 1999a] :

- Pour chaque valeur $a_i \in D(X)$ on définit $l_i = 0$ et $u_i = 1$.
- Pour chaque variable $x \in X$ et pour chaque valeur $a \in D(x)$, $cost(x, a) = \alpha_i a$

Alors, il est facile de prouver que la contrainte $costgcc(X, D(X), l, u, cost, H)$ représente la conjonction des contraintes $\sum_{x_i \in X} \alpha_i x_i \leq H$ et alldiff(X).

Aussi, l'algorithme de filtrage réalisant la consistance d'arc pour la contrainte costGCC réalise également la consistance d'arc pour la contrainte somme et produit scalaire de variables toutes différentes.

On remarquera qu'il est possible de généraliser cette contrainte afin de prendre en compte une contrainte globale de cardinalité au lieu d'une contrainte alldiff.

3.3.4 Contrainte k-diff

Présentation

Cette contrainte est une relaxation de la contrainte alldiff. Au lieu d'imposer que les valeurs prises soient toutes différentes, autrement dit que n valeurs soient prises par n variables, cette contrainte impose qu'un ensemble de variables prennent au moins k valeurs différentes. Cette contrainte est utile lorsque l'on a besoin de relaxer une contrainte alldiff, notamment dans le cas de problèmes sur-contraints. Formellement on a :

Définition 13 *Une* **contrainte k-diff** *est une contrainte C associée à un entier k telle que*

$T(C) = \{\tau$ *où* τ *est un tuple de* $X(C)$ *et*
$|\{a_i \in D(X(C))$ *avec* $\#(a_i, \tau) \geq 1\}| \geq k\}$

Résultats

J'ai introduit cette contrainte dans ma thèse de Doctorat [Régin, 1995]. On a immédiatement la proposition suivante :

Proposition 8 ([Régin, 1995]) *Etant donné C une contrainte k-diff. C est consistante si et seulement si il existe un couplage de taille k dans $GV(C)$.*

Une proposition est alors particulièrement intéressante par rapport à la consistance d'arc :

Proposition 9 ([Régin, 1995]) *Etant donné C une contrainte k-diff. S'il existe un couplage de taille $l > k$ dans $GV(C)$ alors C est arc consistante.*

Considérons une contrainte k-diff consistante et M un couplage de taille k. L'algorithme de filtrage réalisant la consistance d'arc pour cette contrainte recherche donc tout d'abord s'il existe dans $GV(C)$ un couplage de taille strictement supérieur à k (cela se fait aisément à partir de M). Si c'est le cas alors la contrainte est arc-consistance. Sinon, on applique exactement le même algorithme que pour la contrainte alldiff à partir de M.

3.3.5 Contraintes sur des variables ensemblistes

Présentation

Les variables ensemblistes ont été introduites par J-F Puget dans ILOG Solver et par C. Gervet dans Conjunto [Gervet, 1994]. Une variable ensembliste est une variable dont le domaine est constitué d'ensembles. Une variable ensembliste x est associée à deux ensembles connus qui représentent respectivement une borne supérieure, notée $ub(x)$ et inférieure, notée $lb(x)$, au sens de l'inclusion ensembliste et d'une variable de cardinalité, notée $card(x)$ qui définit le cardinal de la variable ensembliste. Par exemple, une variable ensembliste x associée à $lb(x) = \{a\}$, $ub(x) = \{a, b, c, d\}$ et $D(card(x)) = \{1, 2, 3\}$ signifie que l'ensemble auquel x sera affecté contiendra tous les éléments de $lb(x)$, donc l'élément a et sera inclus dans $ub(x)$; sa cardinalité sera égale à $card(x)$. Les différentes affectations possibles pour x seront donc $\{a\}$, $\{a, b\}$, $\{a, c\}$, $\{a, d\}$, $\{a, b, c\}$, $\{a, b, d\}$, $\{a, c, d\}$. On dit souvent que les éléments de $lb(x)$ sont les éléments requis et que les éléments de $ub(x)$ sont les éléments possibles.

Définition 14 *Une contrainte C définie sur des variables ensemblistes est consistante si et seulement si il existe une affectation des variables de C qui satisfait la contrainte et telle que pour toute variable x impliquée dans C, x est affectée avec un ensemble de valeur contenant $lb(x)$ et inclus dans $ub(x)$ et dont la cardinalité est compatible avec $card(x)$.*

Définition 15 *Soit x une variable ensembliste et C une contrainte impliquant x.*

• *L'ensemble $ub(x)$ est consistant avec C si et seulement si pour toute valeur a de $ub(x)$, C admet une solution affectant x avec un ensemble contenant a.*

• *L'ensemble $lb(x)$ est consistant avec C si et seulement si toute solution de C affecte x à un ensemble incluant $lb(x)$ et il n'existe pas d'ensemble incluant strictement $lb(x)$ qui soit inclus dans toutes les affectations de x dans toutes les solutions. En d'autres termes $lb(x)$ est maximal par rapport à l'inclusion.*

Définition 16 *Une contrainte C définie sur des variables ensemblistes vérifie la consistance de borne si et seulement si pour toute variable x, $lb(x)$ et $ub(x)$ sont consistantes avec C.*

Deux contraintes globales sont particulièrement utiles pour les variables ensemblistes : allnullIntersect et partition. La contrainte allnullIntersect définie sur un ensemble X de variables ensemblistes impose que les variables ensemblistes soient deux à deux disjointes. La contrainte partition impose en plus que chaque valeur appartienne exactement à la borne inférieure d'une variable ensembliste. ces contraintes tiennent également compte des cardinalité des différentes variables.

Résultats

Bien que jamais publiés, les algorithmes de filtrage réalisant la consistance de bornes pour ces contraintes sont présents dans ILOG Solver depuis 1997. Je suis l'auteur de ces algorithmes de filtrage.

Ces contraintes se modélisent sous la forme de contraintes globales de cardinalité. Le principe utilisé pour les deux contraintes est le même. Commençons par la contrainte partition. On définit une variable par valeur, que nous appellerons variable duale. Le domaine de ces variables duales correspondant à une valeur a est constitué par les indices des variables ensemblistes qui contiennent a dans l'ensemble des possibles. Si une valeur a appartient à un ensemble de valeurs requises d'une variable ensembliste d'indice i alors la variable duale correspondant à a sera instanciée avec i. Les cardinalités des variables ensemblistes sont prises en compte au travers des bornes de capacités l_i et u_i associées avec un indice i, c'est-à-dire que l'on aura $l_i = min(card(x_i))$ et $u_i = max(card(x_i))$. L'algorithme de filtrage par consistance d'arc de la contrainte globale de cardinalité définie sur les variables duales et associée à $\{l_i\}$ et $\{u_i\}$ réalise la consistance de borne pour la contrainte de partition en liant les variables ensemblistes et les variables duales de telle sorte que la disparition d'une valeur i du domaine d'une variable duale correspondant à a entraîne la suppression de la valeur a de l'ensemble des possible de la variable ensembliste x_i.

Pour la contrainte de partition, chaque valeur doit être prise exactement une fois. Ce n'est plus le cas pour la contrainte allnullIntersect. On modélise alors facilement une contrainte allnullIntersect par une contrainte de partition en introduisant une variable ensembliste supplémentaire dont l'ensemble des possibles contient toutes les valeurs et dont la cardinalité n'est pas borné. On peut alors facilement filtrer par consistance de borne cette contrainte.

3.4 Autres contraintes globales

3.4.1 Contrainte combinant une somme et des inégalités binaires

Présentation

Une contrainte globale `IS` représente la conjonction d'une contrainte de somme et d'un ensemble d'inégalités binaires définies sur les variables impliquées dans la somme. Formellement on a :

Définition 17 *Soient $S_{um}(X, y)$ une contrainte de somme et $\mathcal{I}_{neq}$ un ensemble d'inégalités binaires du type $x_j - x_i \leq c_{ij}$, où x_i et x_j sont des variables et c_{ij} une constante, définies sur $X = (x_1, \ldots, x_r)$. Une contrainte globale $\texttt{IS}(X, y, \mathcal{I}_{neq})$ est définie par l'ensemble des tuples $T(IS)$:*

$$T(IS) = \{ \quad \begin{array}{ll} \tau \text{ où } \tau \text{ est un tuple de } X \cup \{y\}, & \text{et} \\ (\sum_{i=1}^{r} \tau[x_i]) - \tau[y] = 0, & \text{et} \\ \text{les valeurs de } \tau \text{ satisfont } \mathcal{I}_{neq} \ \} & \end{array}$$

Cette contrainte se rencontre dans certains problèmes d'optimisation pour laquelle la fonction objectif est définie par une somme $y = \Sigma x_i$, et les variables x_i sont soumises à des inégalités de la forme $x_j - x_i \leq c_{ij}$. Ce cas se produit par exemple, lorsque l'on ordonne les variables impliquées dans l'objectif afin de casser les symétries. Deux applications importantes de ces contraintes sont la minimisation des retards et le problème de "*minimizing mean flow time*" en ordonnancement déterministe [Blazewicz et al., 1993].

Résultats

J'ai proposé avec M. Rueher [Régin and Rueher, 2000], un algorithme de filtrage qui réalise la **consistance d'intervalle** [van Hentenryck et al., 1998] pour cette contrainte globale. La consistance d'intervalle est dérivée d'une approximation de la consistance d'arc pour les domaines continus. Elle est basée sur une approximation de domaines finis par des ensembles finis d'entiers successifs. Plus précisément, si D est un domaine la consistance d'intervalle considère à la place l'ensemble D^* défini par $[\min(D), \max(D)]$ où $\min(D)$ et $\max(D)$ sont respectivement les valeurs minimum et maximum de D. Une contrainte C vérifie la consistance d'intervalle si et seulement si pour tous les $x_i \in X(C)$ on a $\min(D(x_i)) \leq \max(D(x_i))$ et $\min(D(x_i))$ et $\max(D(x_i))$ sont consistantes avec C, en considérant les domaines D^* au lieu des domaines D.

Considérons une contrainte globale `IS` définie par $\mathcal{I}_{neq} \cup \mathcal{D}_{om} \cup \{S_{um}\}$. Chaque fois qu'une borne de y est modifiée, l'algorithme de filtrage de `IS` réalise les opérations suivantes :

1. Filtrer $\{\mathcal{I}_{neq} \cup \mathcal{D}_{om}\}$ et S_{um} par consistance d'intervalle
2. Modifier les bornes de chaque variable x_i par rapport à la contrainte $\mathcal{I}_{neq} \cup \mathcal{D}_{om} \cup \{S_{um}\}$.

L'étape 1 ne pose pas de problèmes particulier. En effet, la contrainte de somme est une contrainte bien connue et un filtrage par consistance d'intervalle pour les contraintes $\{\mathcal{I}_{neq} \cup \mathcal{D}_{om}\}$ a été proposé par [Dechter et al., 1991]. Cet ensemble de contraintes constitue le problème de la satisfaction de contraintes temporelles simples. La consistance d'intervalle est réalisée en recherchant des plus courts chemins dans un graphe particulier $G = (X, E)$, appelé le graphe des distance, où l'ensemble des nœuds est l'ensemble des variables et l'ensemble E correspond aux inégalités binaires. Comme G peut contenir des cycles négatifs la recherche des plus courts chemins se fait en $O(nm)$ (la présence d'un cycle négatif est une preuve d'inconsistance de la contrainte). La consistance d'intervalle de cette étape peut donc être réalisée en $O(mn)$ où n est le nombre de variables et $m = |\mathcal{I}_{neq}| + 2n$.

L'algorithme donné dans [Régin and Rueher, 2000] permet d'effectuer l'étape 2. Cet algorithme est basé sur la recherche de plus courts chemins et conduit à un algorithme de filtrage par consistance d'intervalle de la contrainte globale `IS` en $On(m + n \log n)$.

Perspectives

La prise en compte d'inégalités ternaires ($x \leq y + z$) mériterait d'être étudiée. L'idée étant d'essayer de considérer une relaxation de ses inégalités ternaires, par exemple en identifiant certaines variables pour ne tenir compte que de la valeur minimale ou maximale de leurs domaines afin de retrouver le type d'inégalité impliquée dans la contrainte IS, puis de réintroduire ces variables dans les inégalités en relachant d'autres variables.

3.4.2 Contrainte globale de séquence

Présentation

Une contrainte globale de séquence (GSC) C est défini sur un ensemble ordonné de variables et est associée à un ensemble V de valeurs, où chaque valeur $a_i \in V$ est associée à deux entiers l_i et u_i, et à des entiers q, min et max. D'une part, elle contraint le nombre de variables affectées à une valeur a_i à appartenir à l'intervalle $[l_i, u_i]$. D'autre part, elle contraint pour chaque séquence S_i de q variables consécutives de $X(C)$, que au moins min et au plus max variables de S_i soient affectées à une valeur de V.

Cette contrainte est très fréquente dans les problèmes d'emploi du temps ou encore dans les problèmes d'ordonnancement de voitures sur une chaîne de montage (car sequencing).

Formellement, on a :

Définition 18 *Une* **contrainte globale de séquence** *est une contrainte C associée à trois entiers positifs min, max, q, une ensemble de valeurs V pour lequel chaque valeur a_i est associée à deux entiers positifs l_i et u_i telle que*

$$\begin{aligned} T(C) = \{ \quad & t \text{ où } t \text{ est un tuple de } X(C) \\ & \text{et } \forall v_i \in V : l_i \leq \#(v_i, t) \leq u_i \\ & \text{et pour chaque séquence } S \text{ de } q \text{ variables} \\ & \text{consécutives : } min \leq \textstyle\sum_{v_i \in V} \#(v_i, t, S) \leq max\} \end{aligned}$$

Résultats

J'ai proposé, avec J-F Puget, un algorithme de filtrage associé à cette contrainte [Régin and Puget, 1997]. Cet algorithme est relativement complexe à comprendre. Aussi, nous allons essayer d'en donner les principes de base.

L'idée principale est assez générale. Il s'agit d'essayer de prendre en compte à la fois les contraintes locales sur chaque séquence et la contrainte globale de cardinalité impliquant les valeurs de V.

Considérons un ensemble de séquences disjointes d'au plus q variables, tel que cet ensemble recouvre $X(C)$. On va construire pour cet ensemble de séquences une contrainte globale de cardinalité qui aura la particularité de prendre en compte les contraintes de séquence pour les séquences considérées et les contraintes de cardinalité sur les valeurs de V. Pour chaque séquence S_i, on définit de nouvelles variables à partir des variables d'origine de la séquence, de façon à ce que si une variable est affectée à une valeur de V alors la nouvelle variable soit également affectée à cette valeur et si une variable n'est pas affectée à une valeur de V alors la nouvelle variable soit affectée à une nouvelle valeur définie pour la séquence. Notons nx une nouvelle variable créée à partir de la variable x et $e(S_i)$ une valeur propre à la séquence S_i et n'appartenant pas à $D(X)$ ni à V. Le domaine de nx est initialement défini par $D(nx) = (D(x) \cap V) \cup e(S_i)$. On définit alors la contrainte ($x = nx$ xor $nx = e(S_i)$) pour chacune des nouvelles variables. Associons deux entiers avec la valeur $e(S_i)$: $l_{e(S_i)} = w(S_i) - max$ et $u_{e(S_i)} = q - min$, où $w(S_i)$ représente la longueur de la séquence. On peut alors définir une contrainte globale de cardinalité : $gcc(NX, W, l, u)$ avec NX l'ensemble des nouvelles variables, $W = V \cup \{e(S_i)\}$, $l = \{l_i\} \cup \{l_{e(S_i)}\}$ et $u = \{u_i\} \cup \{u_{e(S_i)}\}$.

On remarquera qu'il est particulièrement intéressant de prendre le plus possible de séquences de longueur égale à q. On va donc considérer un certain nombre de partitions des variables en séquences de taille au plus q, de façon à ce que toute séquence de longueur q soit membre d'au moins une de ces partitions. Pour chacune de ces partitions on définira alors une contrainte globale de cardinalité. Puis, on introduira de nouvelles contraintes liant entre elles les séquences ne différant que

par une ou deux variables, afin de renforcer les liens entre les différentes contraintes globales de cardinalité.

Grâce à cette contrainte, certains problèmes de la CSP-Lib ont été fermés pour la première fois, notamment ceux concernant le problème de l'ordonnancement de véhicules sur une chaîne de montage. A notre connaissance, cela reste aujourd'hui la seule méthode capable d'obtenir de tels résultats. Plus précisément, cette méthode est la seule qui soit capable de prouver que certains problèmes n'ont pas de solution.

Travaux Connexes

La contrainte globale de distance minimum est proche de la contrainte globale de séquence. J'ai proposé cette contrainte [Régin, 1997] et l'ai introduite dans ILOG Solver[ILOG, 1999].

Une *contrainte globale de distance minimum* définie sur un ensemble X de variables impose que pour chaque paire de variables x et y de X la contrainte $|x-y| \geq k$ soit satisfaite. Formellement on a :

Définition 19 *Une* **contrainte globale de distance minimum** *est une contrainte C associée à un entier k telle que :*

$$T(C) = \{ \quad \tau \text{ où } \tau \text{ est un tuple de } X(C) \text{ et } \forall a_i, aj \in \tau : |a_i - a_j| \geq k\}$$

Cette contrainte est notamment présente dans les problèmes d'allocation de fréquences. On remarquera que si $k = 1$ alors cette contrainte est équivalente à la contrainte alldiff.

L'algorithme de filtrage de cette contrainte utilise l'algorithme de filtrage de la contrainte de séquence. En effet, une contrainte de séquence de type $1/q$, c'est-à-dire imposant que pour toute séquence de q variables consécutives au plus une variable prendra une valeur d'un ensemble V, impose en particulier que deux variables affectées à une valeur de V soient séparée par au moins $q-1$ variables ne prenant pas cette valeur. On utilise donc une représentation duale de la contrainte globale de séquence pour cette contrainte : pour chaque valeur de $min(D(X))$ à $max(D(X))$ on définit une variable, dite variable duale. Notons I l'ensemble des indices des variables d'origine. Une variable duale prend comme valeur l'indice d'une variable d'origine ou bien une valeur particulière ϵ. Une valeur i correspondant à l'indice i appartient au domaine d'une variable duale p si et seulement si $p \in D(x_i)$. A l'origine ϵ appartient au domaine de toutes les variables duales. Il suffit ensuite de définir une contrainte globale de séquence sur cet ensemble de variables duales, impliquant l'ensemble $I \cup \{\epsilon\}$ et imposant que chaque valeur d'indice est prise exactement une fois et qu'au plus une variable d'une séquence de k variables consécutives puisse prendre une valeur de I.

Perspectives

Il serait intéressant de généraliser les principes de la contrainte de séquence afin de voir si l'on peut définir de nouvelles contraintes plus générales. La contrainte de distance minimum mérite également une attention particulière. On pourrait, par exemple, essayer d'introduire une disjunction au lieu de considérer la valeur absolue de la différence entre deux variables, afin de se rapprocher de la problématique de la théorie de l'ordonnancement.

3.5 Perspectives

Pour les contraintes globales on peut envisager au moins deux nouvelles voies de recherche : la définition de contraintes dont le problème sous-jacent est NP-

Complet et l'utilisation d'algorithmes d'approximation pour évaluer la consistance d'une contrainte.

La contrainte de séquence est un exemple de contrainte que l'on qualifie de "NP-Complète". Il en existe d'autres, notamment la contrainte du nombre minimum de valeurs distinctes prisent par un ensemble de variables [Beldiceanu, 2001b]. Cette contrainte s'appuie en fait sur une généralisation du problème Hitting-Set et s'avère particulièrement importante pour résoudre les problèmes de recouvrement. De façon générale on peut imaginer de définir de nombreuses contraintes "NP-Complètes". Toutefois, la définition d'une contrainte n'est intéressante que si l'algorithme de filtrage associé est puissant. On peut alors utiliser des algorithmes d'approximation, comme l'a proposé M. Sellmann [Sellmann, 2003]. Cependant, il faut s'assurer que certaines propriétés comme la monotonie du filtrage ou la stabilité à l'introduction de nouvelles contraintes sont satisfaites. En effet, si l'introduction d'une nouvelle variable ou d'une nouvelle contrainte provoque moins de filtrage alors il sera très difficile d'utiliser une telle contrainte, car la corrections de bugs deviendra très difficile et les modules d'explications ne fonctionneront plus. Cette voie de recherche est donc délicate mais c'est certainement l'une des plus prometteuses.

Chapitre 4

Problèmes sur-contraints

Un problème est sur-contraint quand aucune affectation de valeurs aux variables ne satisfait toutes les contraintes. Dans cette situation, le but est alors de trouver un compromis. Des violations de contraintes sont autorisées à condition que ces violations soient acceptables en pratique. Aussi, il est obligatoire de respecter certaines règles et critères définis par l'utilisateur.

Habituellement, l'ensemble des contraintes initiales est divisé en deux groupes : les contraintes dures, c'est-à-dire celles que l'on ne peut en aucun cas violer, et les contraintes molles, c'est-à-dire les contraintes dont la violation est possible. Un *coût de violation* est généralement associé à chaque contrainte molle. Ensuite, un objectif global impliquant l'ensemble des coûts de violation est défini. Par exemple, le but peut être de minimiser la somme totale des coûts de violations.

Les problèmes sur-contraints sont très fréquents en pratique et les moteurs de résolution à base de PPC sont assez mal adaptés. Aussi, il était nécessaire de regarder de plus près ce type de problème. Je me suis intéressé à ce problème et j'ai travaillé avec mon étudiant en thèse de l'époque (Thierry Petit). Ce travail a été réalisé en grande partie dans le cadre du projet ECSPLAIN (voir description détaillée en première partie de ce mémoire).

Les problèmes sur-contraints ont fait l'objet de nombreuses études. Plusieurs modèles ont été présentés, notamment les "Valued CSP", comme nous le détaillerons par la suite. Cependant cette approche a tout de suite montrée ses limites pour résoudre des problèmes réels.

Nous commencerons par présenter les VCSP et montrerons les limites de ce modèle en étudiant les caractéristiques des problèmes réels sur-contraints. Puis, nous proposerons une méthode qui s'intègre parfaitement dans le cadre de la PPC. En cela nous nous opposons à l'affirmation de G. Verfaillie [Verfaillie, 1997] : "Il est vite apparu que, malgré sa généralité, le formalisme CSP restait un cadre trop restrictif pour capturer toute la complexité des problèmes réels".

Ensuite, nous nous intéresserons à un problème particulier : le problème Max-CSP qui consiste à chercher à trouver une solution minimisant le nombre de contraintes violées. Nous présenterons de façon originale les algorithmes existant et montrerons en quoi nous avons amélioré ces algorithmes. Pour finir nous introduirons le concept de contraintes globales molles et proposerons deux définitions générales des coûts de violation d'une contrainte.

4.1 Méthodes générales

Cette présentation est inspirée de la thése de T. Petit [Petit, 2002].

Deux paradigmes génériques ont été proposés afin d'exprimer toutes les classes de

Classe	$e \in E$	$\oplus$	$\bot$	$\top$	$\succ$
Weighted CSP	$[0, n]$	$+$	0	∞	$>$
CSP possibilistes	$[0, 1]$	max	0	1	$>$
CSP flous	$[0, 1]$	min	1	0	$<$
CSP lexicographiques	$[0, 1]^* \cup \top$	$\cup$	$\varnothing$	$\top$	lexicographique

$[0, 1]^*$ désigne l'ensemble des multi-ensembles (c'est-à-dire des ensembles où l'on peut avoir plusieurs occurrences des éléments) de nombre entre 0 et 1. Il est complété d'un élément spécifique $\top$; $\cup$ est l'union multi-ensembliste étendue pour traiter $\top$ comme un élément absorbant.

FIG. 4.1 – Représentation de différents problèmes à l'aide des VCSP.

problèmes sur-contraints : les CSP valués [Schiex et al., 1995] et les Semi-ring CSPs [Bistarelli et al., 1995]. Leur principe commun est d'associer à chaque contrainte C une valuation dépendante des valeurs affectées aux variables de C. Cette valuation est prise dans un ensemble E ordonné. Un critère d'optimisation est alors défini sur l'ensemble des valuations. Il a été démontré dans [Bistarelli et al., 1999] que ces deux approches ont des propriétés essentiellement équivalentes. Si E est totalement ordonné alors il est même possible de passer d'un CSP valué à un Semi-ring CSP, et vice-versa. Aussi, nous ne décrivons ici que les CSP valués. Pour plus d'informations sur les Semi-ring CSP on pourra consulter [Codognet and Rossi, 2000, Bistarelli et al., 1997, Bistarelli et al., 2002, Dubois et al., 1993].

Dans les CSP valués, lorsque plusieurs contraintes sont violées, la combinaison des valuations est effectuée selon une loi de composition interne $\oplus$. Etant donnés E, $\oplus$ et une relation d'ordre $\succ$, on définit :

Définition 20 *Une structure de valuation est un triplet $(E, \succ, \oplus)$ tel que :*

- *E soit un ensemble totalement ordonné par $\succ$, muni d'un élément minimum noté $\bot$ et un élément maximum noté $\top$.*
- *E soit muni d'une loi de composition interne commutative et associative notée $\oplus$ qui vérifie :*
  - *$\forall a, b, c \in E$ tels que $b \succ c$ on a : $(a \oplus b) \succ (a \oplus c)$*
  - *élément neutre : $\forall a \in E, a \oplus \bot = a$*
  - *élément absorbant : $\forall a \in E, a \oplus \top = \top$*

Définition 21 *Un CSP valué $VCSP = (X, D, \mathcal{C}, S, \varphi)$ est défini par :*

- *un réseau de contraintes $N = (X, D, \mathcal{C})$,*
- *une structure de valuation $S = (E, \succ, \oplus)$,*
- *une application φ de $\mathcal{C}$ dans E associant une valuation à chaque contrainte.*

Ce modèle est plus général que d'autres types de CSPs qui ont été étudié, comme le montre le tableau donné en figure 4.1 qui exprime comment certains types de CSP se représentent facilement dans le cadre des VCSP en fixant certains paramètres :

Nous détaillons brièvement les différents types de problèmes considérés dans le tableau précédent.

Weighted CSP : Dans ce problème [Freuder, 1989, Freuder and Wallace, 1992], à chaque contrainte est associée une valeur constante qui exprime le coût de sa violation (aussi appelé "pénalité" [Schiex et al., 1997]). On peut ainsi favoriser la satisfaction de certaines contraintes par rapport à d'autres. Le problème consiste à trouver une solution minimisant la somme des coûts des contraintes violées.

CSP possibilistes : Dans ce problème [Schiex, 1992] chaque contrainte est associée à un coût réel entre 0 et 1 correspondant à la "priorité" de la contrainte.

Le problème consiste à trouver une instanciation minimisant le maximum des priorités des contraintes violées. Le critère d'optimisation est donc basé sur un opérateur idempotent (c'est-à-dire, $a \oplus a = a$), contrairement au cas des Weighted CSPs. Cette classe de problèmes s'interprète dans le cadre de la théorie des possibilités : si $p(C)$ est associé à une contrainte C, cela signifie que la nécessité que la contrainte soit satisfaite est au moins de $p(C)$.

CSP flous : Ils peuvent être vus comme une extension des CSP possibilistes, tel qu'un coût soit affecté à chaque tuple d'une contrainte, et non plus à la contrainte en elle-même [Dubois et al., 1993]. Ainsi, un tuple de coût 1 est autorisé, alors qu'un tuple de coût 0 viole la contrainte de façon maximale. Le coût d'une contrainte n'est donc plus nécessairement constant : il dépend des valeurs affectées aux variables impliquées. Dans sa version la plus simple, l'objectif consiste a maximiser le coût minimal d'un tuple (correspondant à la violation la plus importante).

CSP lexicographiques : Au lieu d'évaluer la qualité d'une instanciation uniquement en fonction de la valuation de la contrainte violée la plus importante, on prend en compte le nombre de contraintes violées à chacun des niveaux d'importance exprimés. On peut ainsi ordonner les solutions lexicographiquement. Le lecteur intéressé pourra se référer aux travaux présentés dans le cadre des logiques non monotones [Benferhat et al., 1993], et étendus au cadre CSP [Fargier et al., 1993].

4.2 Etude des problèmes réels sur contraints et critique des VCSP

Les CSP valués ont pour vocation de proposer un cadre formel pour modéliser et résoudre les problèmes sur-contraints. Ces modèles sont utilisés par de nombreux chercheurs. De nombreux articles utilisant ce cadre formel ont été publiés. Or, et au risque de surprendre, nous allons montrer que ce cadre formel n'a essentiellement qu'un intérêt théorique, qu'il est assez irréaliste pour modéliser des problèmes réels et, enfin, que ce n'est pas un bon modèle de programmation par contrainte.

Définir une solution d'un problème réel sur-contraint n'est pas une tâche facile. En effet, certaines contraintes doivent être relâchées, autrement dit on accepte une "certaine" violation de ces contraintes. Cela revient bien souvent à définir des règles sur ces relachements. J'ai proposé avec T. Petit et C. Bessière [Petit et al., 2000] d'étudier les règles de relâchement des contraintes les plus fréquentes en pratique :

- **Contraintes inviolables**. Les problèmes réels impliquent toujours des contraintes que l'on ne peut pas violer, notamment des contraintes relatives à la securité des personnes ou des contraintes physiques comme le fait qu'il n'est pas possible d'être en même temps à deux endroits différents.
- **Priorités**. Les contraintes molles d'un problème n'ont pas toutes la même importance, car certaines violations sont moins graves que d'autres. Dans ce cas on essaie de favoriser la satisfaction des contraintes les plus importantes.
- **Quantification de la violation**. Il y a bien souvent plusieurs manières, plus ou moins graves, de violer une contrainte. Par exemple, si une personne doit finir sa journée à 20h alors en terminant sa journée à 20h30 cette personne viole cette contrainte mais de façon moins importante que si elle finit à 22h.
- **Répartition des violations**. Cette contrainte est certainement la plus commune en pratique. En effet, il est bien souvent nécessaire de contrôler la répartition (au sens topologique) des contraintes violées dans le réseau. Par

exemple, il vaut mieux dépasser sur de courtes périodes espacées la charge maximale autorisée sur une machine que de le faire sur une longue période. Il vaut également mieux dans le cas d'emploi du temps que tout le monde ait un emploi du temps à peu près équivalent en terme de respect des souhaits plutôt que certains aient un emploi du temps parfait et d'autres un emploi du temps ne correspondant pas du tout à leurs désirs. On trouve dans la littérature des exemples où, au contraire, on veut concentrer les violations sur des zones précises ; un exemple original est celui des contraintes musicales [Truchet et al., 2001].

- **Règles spécifiques simples indépendantes des valeurs de coût**. Ces règles imposent des relations entre les différentes contraintes violées. Par exemple, on ne peut pas violer certaines contraintes en même temps, ou encore la violation d'une contrainte entraîne l'interdiction de violer une autre contrainte.
- **Règles spécifiques simples dépendantes de certaines valeurs de coût**. Ces règles sont semblables aux précédentes en tenant compte des coûts de violation. Autrement dit, elles définissent de nouvelles contraintes sur les contraintes violées en fonction des coûts des violations. Par exemple, deux contraintes ne peuvent pas être violées en même temps si elles sont fortement violées. En revanche si leur coût de violation est faible alors il est envisageable de les violer simultanément.
- **Règles spécifiques ajoutant de nouvelles contraintes indépendantes des valeurs de coût**. Ces contraintes introduisent de nouvelles contraintes en fonction de la violation d'autres contraintes. Par exemple, on peut imaginer qu'une personne qui doit normalement travailler de 8h à 16h ne doive pas venir travailler avant 10h si la veille elle a travaillé jusqu'à 20h. Dans ce cas on notera qu'une contrainte est introduite dans le problème uniquement si certaines contraintes ont été violées.
- **Règles spécifiques ajoutant de nouvelles contraintes et dépendant de certaines valeurs de coût**. Ces règles sont semblables aux précédentes, excepté que c'est le niveau de violation qui est pris en compte et non plus seulement le fait qu'il y ait une violation ou pas.

Ce qui est surprenant c'est que le modèle des VCSP est incapable de prendre en compte certaines de ces règles. Voici un tableau résumant les capacités de ce cadre formel :

Type de règle	Modèle VCSP
Contraintes inviolables	oui
Priorites	oui
Quantification des violations	oui
Répartition des violations	non
Règles spécifiques simples indépendantes des valeurs de coût	oui
Règles spécifiques simples liées à des valeurs de coût particulières	non
Règles spécifiques ajoutant de nouvelles contraintes et indépendantes des valeurs de coût	oui
Règles spécifiques ajoutant de nouvelles contraintes et liées à des valeurs de coût particulières	non

Ce tableau montre donc certaines limites du modèle VCSP. En fait, il souffre de graves défauts pour son utilisation en PPC. Nous pouvons en présenter trois :

1. **Les coûts sont représentés par une fonction**. Le modèle VCSP propose d'utiliser une application φ liant un tuple d'une contrainte avec un coût de violation (éventuellement nul s'il n'y pas de violation). Il ne s'agit en aucun cas de définir une quelconque contrainte et l'exploitation de la structure de cette application n'est absolument pas mentionné dans le cadre des VCSP. D'ailleurs elle semble difficile à réaliser puisque le modèle n'impose aucune connaissance sur l'existence même de φ^{-1}. Avec le modèle des VCSP il n'est donc pas possible d'éliminer des valeurs des domaines des variables en fonction d'une modification de l'objectif. Or, l'un des principes majeurs de la PPC est la possibilité d'une telle exploitation. Ce principe est appelé "backpropagation" dans ILOG Solver et il est particulièrement important pour la résolution des problèmes d'optimisation.
2. **L'objectif n'est pas une variable**. Il est donc difficile de combiner des problèmes indépendants, puisque l'objectif n'étant pas une variable on ne peut pas définir de contraintes impliquant différents objectifs, ou alors cela demande une extension du modèle des VCSP. Une combinaison se fait en écrivant un nouveau VCSP dont la loi de composition combine les lois associées à chaque problème. La combinaison doit donc se faire de façon intrusive dans les modèles des deux problèmes. Or, cela ne correspond pas à l'idée de base de la PPC qui propose justement d'éviter ce genre d'inconvénient.
3. **La loi de composition interne est limitante**. Avec une telle loi il est impossible d'exprimer de façon raisonnable une règle de contrôle de la répartition des violations dans le réseau. Ce n'est pas étonnant car de telles règles sont par définition globales et non linéaires, et donc difficilement exprimable via un critère global. Or la PPC ne justifie en aucun cas cette limitation. Bien au contraire, l'un des avantages de la PPC est sa capacité à prendre en compte n'importe quel type de contrainte. Il n'y a donc aucune raison de limiter ainsi un modèle qui se veut général.

Cela nous amène en fait à conclure que le modèle des VCSP n'est pas un bon modèle de PPC car il ne permet pas de bénéficier des avantages de la PPC (backpropagation, exploitation de la structure des contraintes, introduction de nouvelles contraintes). C'est pourquoi nous avons proposé un nouveau modèle.

4.3 Un nouveau modèle

L'idée communément admise qui a donné lieu à la proposition du modèle des VCSP, est que les algorithmes de filtrage ne peuvent être utilisés que pour les contraintes qui doivent être satisfaites. En effet, la condition de suppression d'une valeur du domaine d'une variable est liée au fait qu'il est obligatoire de satisfaire la contrainte. Cette condition n'est donc plus applicable lorsque l'on autorise la violation de la contrainte. Or, cela ne signifie pas qu'il n'y a pas d'autres moyens de réduire les domaines. Dans le cas des problèmes sur-contraints on peut par exemple utiliser l'objectif et les coûts associés aux contraintes (ou aux tuples) pour obtenir ce type de résultat. Autrement dit, on peut tirer avantage de la structure des contraintes et de la structure des violations de ces contraintes pour réduire efficacement les domaines des variables.

Pour ce faire, on relie la condition de suppression d'une valeur à la nécessité d'avoir une solution ayant un coût acceptable, au lieu de la relier à la satisfaction des contraintes. Pour obtenir ce résultat on va introduire des variables de coût et définir des nouvelles contraintes intégrant ces variables de coût. On aura ainsi un

modèle qui s'intégrera parfaitement en PPC, puisqu'il n'utilise que les principes de base de la PPC.

Par exemple, considérons la contrainte $x \leq y$. Dans le but de quantifier la violation de cette contrainte, un coût est associé à C. Il est défini comme suit :

− si C est satisfaite alors $cost = 0$.

− si C est violée alors $cost > 0$ et sa valuation est proportionnelle à la différence entre x et y, c'est-à-dire, $cost = x - y$.

On remarquera qu'une telle définition de coût est assez réaliste. En pratique il est rarissime d'énumérer les combinaisons des contraintes pour affecter des coûts particuliers à chaque combinaison sans qu'il y ait une structure derrière cette attribution.

Supposons que $D(x) = [90001, 100000]$ et $D(y) = [0, 200000]$, et que le coût soit contraint à être inférieur ou égal à 5. Alors, soit C est satisfaite et $x - y \leq 0$, soit C est violée et $x - y = cost$ et $cost \leq 5$, ce qui implique $x - y \leq 5$. Aussi, nous pouvons immédiatement déduire que globalement on a $x - y \leq 5$ et par propagation cela entraîne $D(y) = [89996, 200000]$. Une telle déduction est faite directement (i.e. en $O(1)$) en propageant les bornes des variables x, y et $cost$. Une telle propagation, qui exploite la structure d'une contrainte d'inégalité, est bien plus efficace que de considérer les valeurs de façon indépendante. Si l'on ignore la structure de la contrainte, la seule manière de filtrer cette contrainte est d'étudier indépendamment les valeurs en regardant pour chacune le coût des tuples associés. Dans ce cas pour réduire le domaine de $D(y)$ il faudrait effectuer au moins $|D(x)| * 89996 = 899960000$ tests. Cela démontre l'intérêt de l'intégration de coûts dans les contraintes et l'exploitation de la structure des coûts de violation des contraintes.

Notre but est d'avoir la même flexibilité pour les coûts de violation que celle que l'on a avec les variables. La manière la plus naturelle d'obtenir ce résultat est d'inclure les coûts de violation sous la forme de variables dans un nouveau réseau de contraintes[1].

Par souci de clarté, nous considérerons que les valeurs de coût associées à une contrainte sont des entiers positifs. 0 signifie que la contrainte est satisfaite et une valeur strictement positive quantifie l'importance de la violation de la contrainte. Cette considération implique seulement que les valeurs appartiennent à un ensemble totalement ordonné.

Nous proposons de résoudre une nouveau problème d'optimisation dérivé du problème initial [Régin et al., 2000, Régin et al., 2001]. Ce nouveau problème implique le même ensemble de contraintes devant être satisfaites $\mathcal{C}_h$, mais l'ensemble des contraintes molles $\mathcal{C}_s$ est remplacé par un ensemble de contraintes disjonctives, noté $\mathcal{C}_{disj}$. Il existe une bijection de $\mathcal{C}_s$ vers $\mathcal{C}_{disj}$. Chaque disjonction implique une nouvelle variable $cost(C) \in X_{costs}$, qui est utilisée pour exprimer le coût de la violation de la contrainte $C \in \mathcal{C}_s$. On a donc également une bijection de $\mathcal{C}_s$ et X_{costs}. Etant donné $C \in \mathcal{C}_s$, la disjonction impliquant C est la suivante :

$$[C \wedge [cost(C) = 0]] \vee [\bar{C} \wedge [cost(C) > 0]]$$

$\bar{C}$ est la contrainte qui définit le coût de violation de C en affectant la variable $cost(C)$. Un algorithme de filtrage spécifique peut être associé à cette contrainte comme à n'importe quelle autre contrainte. Si l'on reprend l'exemple précédant, les contraintes C et $\bar{C}$ sont respectivement $x \leq y$ et $cost(C) = x - y$, on aura alors la disjonction

$$[[x \leq y] \wedge [cost(C) = 0]] \vee [[cost(C) = x - y] \wedge [cost(C) > 0]]$$

[1]Nous devons mentionner que cette inclusion n'est pas toujours facile à effectuer. Ce point est discuter lors de l'etude des contraintes globales molles.

En remplaçant les contraintes $\mathcal{C}_s$ par les contraintes $\mathcal{C}_{disj}$ on obtient un nouveau problème qui n'est plus sur-contraint. Il s'agit alors de satisfaire toutes les contraintes de l'ensemble $\mathcal{C}_h \cup \mathcal{C}_{disj}$, tout en optimisant un objectif défini sur les variables de X_{costs}. Cet objectif est modélisé à l'aide d'une variable objectif et de contraintes liant cette variable aux variables de X_{costs}. N'importe quelle contrainte peut être utilisée, et donc pas seulement une contrainte linéaire.

Un tel modèle peut être utilisé pour coder directement des problèmes sur-contraints dans n'importe quel moteur de résolution basé sur la PPC. Comme l'objectif est une variable liée aux variables de coût des contraintes on peut facilement combiner plusieurs problèmes sur-contraints, de la même façon que l'on combine habituellement des problèmes en PPC.

Ce nouveau paradigme permet d'exprimer l'ensemble des règles de violations présentées précédemment [Petit et al., 2000] et ne présente aucun des inconvénients des VCSP.

4.3.1 Comparaison avec les VCSP

Nous voulons répondre immédiatement à l'objection qui a été faite à l'introduction d'un nouveau modèle et à notre critique des VCSP. Cette objection consiste à prétendre que les VCSP sont un cadre formel et que leur utilisation en PPC demande une adaptation normale et que notre modèle n'est finalement qu'une adaptation des VCSP. Tout d'abord, cela ne correspond absolument pas à la façon dont les VCSP sont présentés. Ensuite, les VCSP ont vocation à proposer un modèle général pour résoudre les problèmes sur-contraintes. Or, il n'est jamais mentionné dans les VCSP que l'opération inverse φ^{-1} est disponible. Par ailleurs, on parle ici de modèle et de puissance de modèle. Certaines contraintes ne peuvent pas être modélisées à l'aide des VCSP, parce que les VCSP n'introduisent pas explicitement des variables de coût et des contraintes impliquant ces variables. Aucun article sur les VCSP n'a jamais proposé ni même envisagé de faire cela. La preuve est que l'introduction de nouveaux objectifs à optimiser à donner lieu à chaque fois à une nouvelle définition de CSP, ce qui n'est pas raisonnable et montre la faiblesse de chaque modèle proposé. Lorsque l'on commence à introduire ces concepts on ne peut donc plus dire que l'on fait des VCSP.

Nous nous sommes aussi efforcé de présenter un modèle plus compréhensible et plus appréhendable que les VCSP. En proposant une structure algébrique, le risque est alors de se concentrer sur cette structure et non plus sur la résolution. Le discours devient alors hermétique pour le novice. Par exemple, on peut dire que le modèle des VCSP est équivalent à une co-norme triangulaire ou à un monoïde conjonctif [Verfaillie, 1997].

Le fait que certains chercheurs à l'origine de l'approche VCSP se soient rangés à notre avis en publiant une adaptation de notre méthode en CHOCO [Lemaitre et al., 2001], nous conforte dans l'idée que cette voie de recherche est prometteuse et mérite d'être approfondie.

4.4 Minimisation du nombre de contraintes violées

4.4.1 Présentation

L'un des problème qui a le plus attiré l'attention des chercheurs travaillant sur les problèmes sur-contraints est celui de la minimisation du nombre de contraintes violées (Max-CSP).

Plusieurs algorithmes ont été proposé pour résoudre ce problème. Tout d'abord Partial Forward Checking [Freuder and Wallace, 1992], qui a été amélioré par PFC-

DAC [Wallace, 1994, Larrosa and P.Meseguer, 1996], puis par PFC-MRDAC [Larrosa et al., 1998]. Ce dernier algorithme peut être également vu comme une généralisation de la disjonction constructive au cas où plusieurs contraintes doivent être satisfaites (et non pas au moins une comme pour la disjonction constructive).

Tous ces algorithmes sont des algorithmes ad-hoc qui sont basés sur l'algorithme Branch-And-Bound. Par ailleurs ils ne considèrent que des contraintes binaires données en extension. La modification de ces algorithmes afin de pouvoir être intégrés dans un solver n'est donc pas simple.

Je propose dans cette section une présentation originale des principes de l'algorithme PFC-MRDAC. Cette présentation, à mon avis, est plus simple que toutes celles proposées jusqu'alors.

Considérons un réseau de contraintes $P = (X, D, \mathcal{C})$, et les notations suivantes :

Notation 1

- $v^*(P)$ *est le nombre minimum de contraintes qui sont violées par* P
- $v(P)$ *désigne n'importe quelle borne inférieure de* $v^*(P)$
- $v^*((x,a),P)$ *est le nombre minimum de contraintes qui sont violées par* P *quand* $x = a$
- $v((x,a),P)$ *désigne n'importe quelle borne inférieure de* $v^*((x,a),P)$

Définition 22 *Deux sous-problèmes* $Q1 = (X, \mathcal{D}, \mathcal{K})$ *et* $Q2 = (X, \mathcal{D}, \mathcal{L})$ *de* P *sont disjoints pour les contraintes si et seulement si* $\mathcal{K} \cap \mathcal{L} = \varnothing$

Théorème 2 *Soient* $P = (X, \mathcal{D}, \mathcal{C})$ *un réseau de contraintes, et* $\mathcal{Q}$ *un ensemble de sous problèmes de* P *deux à deux disjoints pour les contraintes, alors :*

$$v^*(P) \geq \sum_{Q \in \mathcal{Q}} v^*(Q) \geq \sum_{Q \in \mathcal{Q}} v(Q)$$

La preuve est immédiate puisque les contraintes des sous-problèmes sont incluses dans celles de P et sont deux à deux disjointes.

On peut alors écrire deux corollaires de ce théorème :

Corollaire 4 *Soit obj une valeur. Si* $\sum_{Q \in \mathcal{Q}} v(Q) > obj$ *alors il n'existe pas de solutions de* P *avec* $v^*(P) \leq obj$

Corollaire 5 *Soit obj une valeur et a la valeur d'une variable x impliquée dans un sous-problème* Q *de* $\mathcal{Q}$*. Si* $\sum_{R \in (\mathcal{Q}-Q)} v(R) + v((x,a),Q) > obj$ *alors il n'existe pas de solutions de* P *avec* $v^*((x,a),P) \leq obj$

Si *obj* est la variable d'objectif alors le premier corollaire va permettre de couper les branches de l'arbre de recherche et le second de supprimer des valeurs qui ne sont pas consistantes avec la branche courante.

Tout le problème est alors la détermination de l'ensemble $\mathcal{Q}$ et le choix de $v(Q)$ pour un problème Q donné.

L'algorithme PFC-MRDAC propose de construire l'ensemble $\mathcal{Q}$ de la façon suivante : on commence avec l'ensemble de contraintes $\mathcal{K} = \mathcal{C}$ et on ordonne les variables, puis on sélectionne tour à tour chacune des variables selon cet ordre, on prend alors toutes les contraintes de $\mathcal{K}$ impliquant la variable sélectionnée afin de former un sous-problème et on supprime de $\mathcal{K}$ ces contraintes avant de considérer la variable suivante dans l'ordre. Nous noterons $Q(x)$ le sous problème obtenu à partir de la variable x.

La construction particulière de $\mathcal{Q}$ permet d'obtenir une valeur de $v(Q)$ facile à calculer. Chaque sous-problème est défini à partir d'une variable qui est impliquée dans toutes les contraintes de ce sous-problème. On calcule alors pour chaque valeur

de cette variable le nombre des contraintes qui sont violées par cette valeur et l'on définit $v(Q)$ comme étant le minimum de violations directes des valeurs du domaine. Pour effectuer ces calculs rapidement, PFC-MRDAC calcule de façon incrémentale pour chaque variable x et pour chaque valeur a de x la valeur $v((x,a),Q(x))$ qui compte le nombre de contraintes de $Q(x)$ avec lesquelles (x,a) est inconsistante.

Le point de vue que nous venons de donner est très général et devrait permettre l'élaboration de nouveaux algorithmes.

J'ai proposé avec T. Petit, C. Bessière et J-F. Puget d'utiliser le nouveau modèle pour les problèmes Max-CSP [Régin et al., 2000] ainsi que plusieurs améliorations de l'algorithme PFC-MRDAC [Régin et al., 2001, Petit et al., 2002] et enfin un nouvel algorithme [Régin et al., 2001].

4.4.2 Contrainte de somme des satisfactions

Au lieu de proposer un algorithme ad-hoc pour le problème Max-CSP, comme c'est le cas des algorithmes PFC-MRDAC, nous proposons de définir une contrainte représentant ce problème. Cette contrainte utilise les mêmes principes que le nouveau modèle que nous avons présenté. Elle est appelée contrainte de somme des satisfactions.

Définition 23 *Etant donnés $\mathcal{C} = \{C_i, i \in \{1,\ldots,m\}\}$ un ensemble de contraintes, $cost(\mathcal{C})$ l'ensemble des variables de coût associées aux contraintes de $\mathcal{C}$ et unsat une variable. Une* **contrainte de somme des satisfactions** *est une contrainte $ssc(\mathcal{C}, cost(\mathcal{C}), unsat)$ définie par la conjonction de la contrainte*

$$unsat = \sum_{C \in \mathcal{C}} cost(C)$$

et de l'ensemble des contraintes disjonctives

$$\{[(C \wedge (cost(C) = 0)) \vee (\overline{C} \wedge (cost(C) = 1))], C \in \mathcal{C}\}$$

Les variables de $cost(\mathcal{C})$ sont définies dans la section 4.3. La variable *unsat* est une variable d'objectif qui est égale au nombre de contraintes violées dans $\mathcal{C}$, c'est-à-dire à la somme des variables de coût de ces contraintes.

Une solution du problème Max-CSP est une affectation des variables de coût qui satisfait la contrainte de somme des satisfaction et qui minimise la valeur de la variable *unsat*.

Une borne inférieure de l'objectif de Max-CSP correspond à une condition nécessaire pour la consistance d'une contrainte de somme des satisfactions. Les différents algorithmes de réduction de domaines écrits pour Max-CSP deviennent des algorithmes de filtrage associés à cette contrainte.

Ce point de vue a plusieurs avantages par rapport aux études précédentes :

- N'importe quel algorithme de recherche de solutions peut-être utilisé. Comme nous proposons de définir une contrainte nous pouvons facilement intégrer cette contrainte dans n'importe quel moteur de PPC. Cette contrainte peut être combinée avec d'autres contraintes afin de séparer les contraintes violables de celles qui doivent être satisfaites.
- Aucune hypothèse n'est faite sur l'arité des contraintes.
- Si une variable de coût est affectée alors cela signifie que la contrainte (dans le cas où elle vaut 0) ou sa négation (dans le cas où elle vaut 1) doit être satisfaite. Dans ce cas on peut immédiatement utiliser les algorithmes de filtrage associés à ces contraintes, comme on le fait pour n'importe quelle contrainte devant être satisfaite.

4.4.3 Adaptation de PFC-MRDAC aux problèmes dont les variables sont des intervalles

L'algorithme PFC-MRDAC, pour être performant, maintient la valeur d'un compteur du nombre de violations pour chaque valeur de chaque variable. Précisément PFC-MRDAC maintient la valeur de $v((x,a),Q(x))$ pour chaque (x,a) afin de calculer $v(Q(x)) = \min_{a \in D(x)}(v((x,a),Q(x))$.

Or, certains problèmes comme les problèmes de scheduling présentent deux particularités :

• les domaines de certaines variables sont très grands. Par exemple si l'on doit ordonner des activités sur une période d'un an avec une granularité d'un quart d'heure alors on aura un domaine contenant plus de 30000 valeurs.

• de nombreuses contraintes sont associées à des algorithmes de filtrage qui ne filtrent que les bornes. C'est par exemple le cas d'une contrainte $x < y$

Pour ces problèmes il est difficile d'appliquer l'algorithme PFC-MRDAC à cause de sa consommation mémoire.

J'ai proposé avec T. Petit et C. Bessière un algorithme permettant d'adapter PFC-MRDAC à ce type de problème [Petit et al., 2002]. L'idée consiste à ne travailler qu'avec les bornes des domaines. Ce nouvel algorithme appelé Range-PFC-MRDAC est basé sur la notion suivante :

Définition 24 *Soit $I \subseteq D(x)$ un intervalle de valeurs consécutives. Si $\forall a \in I, \forall b \in I$, $v((x,a),Q(x)) = v((x,b),Q(x))$ alors I est homogène-inconsistant.*

Toutes les valeurs d'un intervalle homogène consistant violent le même nombre de contrainte, donc on peut considérer l'intervalle en lui-même pour calculer $v(Q(x))$ plutôt que de considérer toutes les valeurs de l'intervalle. $v(Q(x))$ pourra donc être calculé à partir de tels intervalles plutôt qu'à partir des valeurs du domaine.

Notons $v(I,Q(x))$ le nombre de contraintes de $Q(x)$ violées par une des valeurs de I et $\mathcal{I}(Q(x))$ un ensemble d'intervalles homogène-inconsistants qui couvre $D(x)$. On a alors immédiatement la propriété suivante :

Propriété 5 $v(Q(x)) = \min_{I \in \mathcal{I}(Q(x))}(v(I,Q(x))$.

Si on peut calculer rapidement $v(I,Q(x))$ et identifier un ensemble $\mathcal{I}(Q(x))$ dont la cardinalité est nettement plus petite que celle du domaine alors on peut améliorer PFC-MRDAC puisque les calculs de $v(Q(x))$ se feront plus rapidement.

Notons $\mathcal{C}(Q(x))$ l'ensemble des contraintes du problème $Q(x)$. Nous avons montré que si l'on considère uniquement les modifications des bornes des domaines par les algorithmes de filtrage associés aux contraintes alors :

• la taille de $\mathcal{I}(Q(x))$ est au plus $2|\mathcal{C}(Q(x))| + 1$,

• l'ensemble de tous les $v(I,P(x))$ peut être calculé en $O(|\mathcal{C}(Q(x))|)$ à condition d'avoir préalablement calculé $\mathcal{I}(Q(x))$.

On obtient donc un algorithme dont la complexité ne dépend plus des domaines mais du nombre de contraintes. Le but recherché est donc atteint pour les problèmes impliquant un très grand nombre de valeurs. De plus la restriction faite en ne considérant que les modifications des bornes des domaines, n'en est pas vraiment une pour de nombreux problèmes d'ordonnancement puisque la plupart des algorithmes de filtrage ne réduisent que ces bornes.

4.4.4 Contraintes ignorées par PFC-MRDAC

Afin d'améliorer l'algorithme PFC-MRDAC, il est intéressant d'essayer d'appréhender ses limites. Notamment, par l'identification de contraintes dont la suppres-

sion ne change pas le résultat de l'algorithme PFC-MRDAC. C'est le propos de cette section. Pour plus d'informations on pourra consulter [Régin et al., 2001].

Reprenons le formalisme que nous avons utilisé pour présenter PFC-MRDAC.

Définition 25 *Une contrainte C impliquée dans le problème Q est ignorée par une borne inférieure du nombre de violation si $v(Q) = v(Q - \{C\})$.*

Déterminer les contraintes ignorées n'est donc pas une tâche difficile et on peut supprimer une contrainte ignorée sans changer les bornes calculées. Malheureusement ce n'est plus vrai pour deux contraintes ignorées pour le même sous problème Q. Il n' y aucune raison pour que l'on est simultanément $v(Q) = v(Q - \{C1\})$, $v(Q) = v(Q - \{C2\})$ et $v(Q) = v(Q - \{C1, C2\})$.

La définition précédente doit donc être raffinée lorsque l'on veut considérer plusieurs contraintes ignorées à la fois :

Définition 26 *Un ensemble S de contraintes impliquées dans le problème Q est un* **ensemble indépendant de contraintes ignorées** *par une borne inférieure du nombre de violation si $v(Q) = v(Q - S)$.*

Le problème est alors d'être capable de déterminer un tel ensemble de contraintes. On remarquera que si l'on trouve un ensemble indépendant de contraintes ignorées pour chaque sous-problème alors l'union de ces ensembles forme un ensemble indépendant de contraintes ignorées de P, puisque les sous-problèmes sont deux à deux disjoints pour les contraintes.

Il existe un exemple simple d'ensemble indépendant de contraintes ignorées qui mérite d'être mentionné : c'est l'ensemble S de toutes les contraintes d'un sous-problème Q qui vérifie $v(Q) = 0$. En effet, quoique l'on fasse on ne pourra pas diminuer la valeur de cette borne inférieure.

Trouver les plus grands ensembles indépendants de contraintes ignorées est un problème NP-Complet (correspondant à un problème de recouvrement minimal). Nous proposons néanmoins une méthode plus faible pour identifier un ensemble de contraintes ignorées qui soit indépendant. L'algorithme 4.1 est un algorithme glouton qui retourne un sous-ensemble indépendant S de contraintes ignorées de Q.

```
seekIndependentSet(Q : problem ) : ens. de contraintes
K ← C(Q)
S ← ∅
while ∃C ∈ K avec v(Q − (S ∪ {C})) ≥ v(Q) do
    S ← S ∪ {C}
    K ← K − {C}
return S
```

Algorithm 4.1 – Calcul d'un ensemble indépendant de contraintes ignorées.

4.4.5 Utilisation des algorithmes de filtrage associés aux contraintes

Il est regrettable que les filtrages associés aux contraintes ne soient pas utilisés par l'algorithme PFC-MRDAC tel que décrit dans la littérature. Ce dernier ne considère, en effet, que les violations directes, en testant explicitement la consistance de chaque valeur.

Une adaptation de cet algorithme afin de pouvoir directement utiliser les algorithmes de filtrage était donc nécessaire. C'est ce que j'ai proposé avec T. Petit et J-F Puget [Régin et al., 2002]. Dans cet article, nous montrons comment on peut

parvenir à un tel résultat et comment on peut réaliser cela de façon incrémentale bien que les contraintes puissent être violées. Nous invitons le lecteur à consulter cet article pour plus de détails.

4.4.6 Nouvel algorithme de filtrage

Les choix faits par l'algorithme PFC-MRDAC pour l'ensemble des sous-problèmes deux à deux disjoints pour les contraintes et pour le calcul d'un minorant du nombre de violations s'avèrent peu performants lorsqu'il y a peu de violations. Cela se présente par exemple au début de la recherche.

Par exemple, l'algorithme PFC-MRDAC est incapable de détecter une violation pour le réseau impliquant les variables x, y, z avec $D(x) = D(y) = D(z) = \{1, 2, 3\}$ et les contraintes $x > y$, $y < z$ et $z < x$. Cela constitue, à notre avis, une faiblesse importante de l'algorithme. En fait, comme on l'a vu précédemment, il suffit qu'une variable x à partir de laquelle le sous-problème $Q(x)$ est déterminé, contienne une valeur compatible avec toutes les contraintes pour que $Q(x)$ soit en fait totalement ignoré par PFC-MRDAC. Dans l'exemple précédent c'est la cas avec toutes les valeurs 2 de tous les domaines, donc quel que soit la manière de choisir les sous-problèmes PFC-MRDAC ne détectera aucune violation.

Aussi, il est apparu indispensable de proposer un nouvel algorithme qui aurait au moins la capacité de détecter une inconsistance dans ce réseau simple. J'ai proposé avec T. Petit, C. Bessière et J-F. Puget un tel algorithme [Régin et al., 2001].

Cet algorithme est basé sur la notion d'ensemble de conflits ("conflict-set" en anglais).

Définition 27 *Considérons $v(P)$ une borne inférieure du nombre de contraintes de P violées. On appelle* **ensemble de conflits** *par rapport à v, un ensemble K de contraintes de P tel que : $v(K) > 0$.*

Afin de bénéficier des avantages de la PPC (filtrage et propagation) nous proposons de détecter les ensembles de conflits en recherchant simplement si la propagation détecte une inconsistance pour le sous-problème considéré. Si c'est le cas, alors les contraintes de ce sous-problème forment un ensemble de conflits. On remarquera que cette méthode détecte bien une inconsistance avec l'exemple précédent.

Nous pouvons maintenant décrire comment les sous-problèmes du théorème 2 vont être choisis. Nous rappelons que ce théorème établit que pour tout ensemble $\mathcal{Q}$ de sous problèmes de P deux à deux disjoints pour les contraintes, on a $v^*(P) \geq \sum_{Q \in \mathcal{Q}} v^*(Q) \geq \sum_{Q \in \mathcal{Q}} v(Q)$.

On initialise un ensemble K de contraintes avec toutes les contraintes du problème. Puis on applique récursivement la procédure suivante : on crée un problème Q ne contenant aucune contrainte et on introduit une à une les contraintes de K, après chaque addition de contrainte on vérifie si la propagation détecte une inconsistance. Si c'est le cas, alors les contraintes de Q forment un ensemble de conflits, on supprime alors ces contraintes de K et on applique de nouveau la procédure. Sinon, on introduit une nouvelle contrainte. Si toutes les contraintes ont été introduites alors cela signifie que l'on n'est plus capable de détecter facilement une inconsistance et l'algorithme s'arrête. En procédant ainsi on s'assure que les problèmes vont être deux à deux disjoints pour les contraintes et que chaque sous-problème, sauf éventuellement le dernier, vérifent $v(Q) \geq 1$.

Cette méthode est donc une nouvelle méthode permettant de calculer $v(P)$. Elle peut être raffinée en essayant d'engendrer des ensembles de conflits plus petits. Nous avons montré que l'on peut engendrer des ensembles minimaux au sens de l'inclusion, c'est-à-dire des ensembles de contraintes détectés comme étant inconsistants et tel que si on supprime une contrainte alors cet ensemble n'est plus détecté comme étant inconsistant par la propagation.

On peut alors soit utiliser cette méthode toute seule, soit la combiner avec PFC-MRDAC, ce qui va permettre d'améliorer l'un des deux algorithmes. Deux types de combinaisons sont possibles :

1. On commence par déterminer une borne inférieure du nombre de violations avec la méthode des ensembles de conflits, puis on applique l'algorithme PFC-MRDAC sur les contraintes qui n'appartiennent à aucun ensemble de conflits. En procédant ainsi on va améliorer la borne trouvée par la méthode des ensembles de conflits et on va pouvoir en plus filtrer les domaines grâce à PFC-MRDAC.
2. On commence par appliquer l'algorithme PFC-MRDAC, puis on calcule un ensemble indépendant de contraintes ignorées par PFC-MRDAC et on applique la méthode des ensembles de conflits sur cet ensemble. En procédant ainsi on améliore donc l'algorithme PFC-MRDAC.

Mentionnons également le fait que nous avons également proposé une méthode implémentant de façon incrémentale la méthode des ensembles de conflits.

Pour finir cette section, nous montrons que la méthode des ensembles de conflits peut aussi être utilisée comme algorithme de filtrage des domaines, et donc pas seulement pour calculer une borne inférieure globale du nombre de violations. Cette présentation est originale et assez simple grâce au nouveau point de vue que nous avons donné.

Le corollaire 5 est tout à fait applicable avec notre algorithme. Il suffit alors de considérer que $v((x,a),Q)$ compte le nombre de violations directes entraînées par la valeur (x,a) pour Q. On peut même établir un nouveau corollaire, ce qui permet d'introduire un peu d'originalité dans ce document :

Corollaire 6 *Soient*

• $P = (X, \mathcal{D}, \mathcal{C})$ *un réseau de contraintes,*

• $\mathcal{Q}$ *un ensemble de sous problèmes de* P *deux à deux disjoints pour les contraintes, tel que les contraintes de chaque sous-problème forment un ensemble de conflits*

• x *une variable et* $\mathcal{Q}(x)$ *l'ensemble des sous-problèmes de* $\mathcal{Q}$ *qui contiennent une contrainte impliquant* x*.*

• *obj une variable*

• a *une valeur de* x

Si

$$\sum_{Q \in (\mathcal{Q}(x))} \max(1, v((x,a),Q)) + \sum_{R \in (\mathcal{Q} - \mathcal{Q}(x))} v(R)) > obj$$

alors il n'existe pas de solutions de P *avec* $v^*((x,a),P) \leq obj$

4.4.7 Décomposition en parties non disjointes

La présentation qui suit est originale, même si elle s'appuie sur le travail que j'ai effectué avec T. Petit et C. Bessière [Petit et al., 2003a, Petit et al., 2003b].

Le théorème 2 impose que les sous-problèmes considérés soient deux à deux disjoints pour les contraintes afin de pouvoir sommer les violations de chacun des sous-problèmes.

Cette condition est forte, il est donc tentant d'essayer de la relaxer. En fait, dans certains cas on peut calculer en temps polynomial le nombre de violations d'un ensemble de sous-problèmes non disjoints pour les contraintes.

Considérons $P = (X, \mathcal{D}, \mathcal{C})$ un réseau de contraintes, et $\mathcal{Q}$ un ensemble de sous problèmes de P. Notre but est d'exprimer le nombre de contraintes violées pour P en fonction du nombre de contraintes violées pour chacun des sous-problèmes de $\mathcal{Q}$. La difficulté est d'éviter de compter la violation d'une contrainte plusieurs fois.

Le problème auquel nous sommes confronté peut être exprimé de la façon suivante :
Etant donnée une collection de sous-ensembles d'un ensemble E, chaque sous-ensemble S étant associé avec un entier $v(S)$, le but est de trouver un ensemble $E' \subseteq E$ de taille minimale telle que chaque sous-ensemble S de la collection ait $v(S)$ représentant dans E'. Dans notre cas la cardinalité de E' est la valeur que nous recherchons.

Ce problème est plus général que le problème Hitting-Set pour lequel tous les sous-ensembles S vérifient $v(S) = 1$. Hitting-Set est malheureusement NP-Complet [Garey and Johnson, 1979], donc notre problème aussi. Néanmoins il est polynomial dans certains cas. Par exemple, si tous les sous-ensembles sont disjoints, alors la cardinalité de E' est égale à la somme des v, et on retrouve notre résultat.

Pour la méthode des ensembles de conflits nous avons par définition $v(Q) \leq 1$ pour tout sous-problème Q. On retrouve donc exactement le problème Hitting-Set. Hitting Set est NP-Complet même si les sous-ensembles sont de taille au plus 2. Heureusement, Hitting-Set est polynomial dans le cas où un élément (une contrainte dans notre cas) n'appartient pas à plus de deux sous-ensembles (sous-problèmes dans notre cas). Le problème se ramène alors au problème Edge-Cover, c'est-à-dire à la recherche du nombre minimal d'arêtes nécessaire pour couvrir tous les nœuds d'un graphe. Le graphe considéré a pour ensemble de nœuds les sous-ensembles de la collection et il existe une arête entre deux sous-ensembles si et seulement si l'intersection de ces sous-ensembles n'est pas vide. Edge-Cover se résout facilement en cherchant un couplage maximum dans ce graphe. La taille minimale d'un recouvrement des nœuds par des arêtes est égale au nombre de noeuds diminué de la taille du couplage maximum [Berge, 1970].

On peut donc calculer en temps polynomial un minorant du nombre de violations des contrainte en permettant que les sous-problèmes ne soient pas nécessairement disjoints pour les contraintes, mais à la condition qu'une contrainte n'appartienne pas à plus de deux sous-problèmes.

On peut aussi accepter de faire une erreur lors de ce calcul du minorant, en utilisant un minorant du problème HittingSet. Le lecteur intéressé trouvera plus d'informations dans [Petit et al., 2003a, Petit et al., 2003b].

4.5 Contraintes globales molles

Dans le but de pouvoir prendre en compte les problèmes sur-contraints dans un moteur de PPC industriel, j'ai essayé de traduire dans ce domaine ce qui fait le succès de la PPC pour résoudre des problèmes de satisfaction classiques : l'utilisation de la structure des contraintes et la définition de contraintes globales.

Comme les contraintes pouvant être violées intègrent la notion de coût il est intéressant pour pouvoir réutiliser les résultats existants, d'essayer de définir les coûts de façon générale. Si l'on parvient à ce résultat alors on pourra proposer l'équivalent des contraintes globales dans le cas où les violations sont autorisées.

4.5.1 Définitions générales du coût

Lorsque le coût de violation d'une contrainte a une structure qui est explicitement donné, alors on peut, bien sûr, utiliser cette structure pour définir lee coût de violation. Nous avons donné un exemple pour la contrainte $C : x \leq y$. Lorsque le coût de violation pour une contrainte n'est pas structuré de façon explicite, différentes définitions générales peuvent être envisagées selon le problème.

Par exemple, soit C une contrainte alldiff définie sur $\{x_1, x_2, x_3, x_4\}$ et telle que $\forall i \in [1, 4], D(x_i) = \{a, b, c, d\}$. Si nous ignorons les cas symétriques en considérant

qu'aucune valeur n'a plus d'importance qu'une autre et qu'il en est de même pour les variables, alors nous avons les affectations possibles suivantes : (a, b, c, d), (a, a, c, d), (a, a, c, c), (a, a, a, c), (a, a, a, a).

Intuitivement, on a envie de dire que la violation (a, a, a, a) est plus grave que la violation (a, a, c, d). On peut exprimer cette idée au travers d'une première définition générale du coût de violation.

Définition 28 ([Petit et al., 2001]) *Soit C une contrainte. La variable de coût cost qui définit la violation de C comme étant le nombre d'affectation qu'il faut changer pour satisfaire C est appelée* **variable de coût basé sur les variables**.

L'avantage de cette définition est qu'elle peut être appliquée à n'importe quel type de contrainte. Cependant, selon l'application, cette définition peut être insuffisante. Pour la contrainte alldiff de l'exemple précédent nous obtiendrons : $cost((a, b, c, d)) = 0$, $cost((a, a, c, d)) = 1$, $cost((a, a, c, c)) = 2$, $cost((a, a, a, c)) = 2$, et $cost((a, a, a, a)) = 3$. On peut constater que les affectations (a, a, c, c) et (a, a, a, c) ont le même coût selon la définition 28, cela peut être génant pour certaines applications. Pour une contrainte alldiff impliquant plus de quatre variables, de nombreuses affectations auront le même coût.

Aussi, nous avons proposé une autre définition générale du coût qui est basée sur le graphe primal [Dechter, 1992] :

Définition 29 *Le* **graphe primal** *$Primal(C) = (X(C), E)$ d'une contrainte C est le graphe tel que chaque arête représente une contrainte binaire et l'ensemble des solutions du réseau de contrainte $\mathcal{N} = (X(C), D(X(C)), E)$ est l'ensemble des tuples de C.*

Pour une contrainte alldiff C, $Primal(C)$ est le graphe complet dont chaque arête est un contrainte binaire de différence.

Définition 30 ([Petit et al., 2001]) *Soit C une contrainte représentable par son graphe primal. La variable de coût cost qui défini la violation de C comme étant le nombre de contraintes binaires de $Primal(C)$ violées est appelée* **variable de coût basé sur le graphe primal**.

L'avantage de cette définition est qu'elle quantifie de façon plus fine les violations. Avec notre exemple, on obtient : $cost((a, b, c, d)) = 0$, $cost((a, a, c, d)) = 1$, $cost((a, a, c, c)) = 2$, $cost((a, a, a, c)) = 3$ et $cost((a, a, a, a)) = 6$.

4.5.2 Contrainte Alldiff molle

Comme exemple de contrainte globale prenant en compte les coûts de violations que l'on peut obtenir à partir d'une contrainte globale classique, j'ai étudié avec T. Petit et C. Bessière la contrainte "soft alldiff" [Petit et al., 2001].

La contrainte obtenue en combinant le coût de violation basé sur les variables et une contrainte alldiff est , en fait, une contrainte k-diff, où k est la valeur minimum de la variable de coût. lorsque k est modifié (il ne peut être qu'augmenter) on peut facilement modifier la contrainte k-diff et on obtient ainsi un algorithme de filtrage réalisant la consistance d'arc pour cette contrainte globale molle.

La contrainte combinant le coût de violation basé sur le graphe primal et une contrainte alldiff est plus complexe. J'ai écrit un algorithme spécifique pour cette contrainte [Petit et al., 2002]. Sa complexité est en $O(n^2\sqrt{n}Kd)$, où $K = \sum |D(x)|$, $n = |X(C)|, x \in X(C)$ et $d = max(|D(x)|), x \in X(C)$. Un autre algorithme plus intelligent a récemment été proposé par [Jan van Hoeve, 2004].

4.5.3 Perspectives

Nous avons proposé deux définitions générales de coût de violation. Il en existe certainement d'autres. D'ailleurs, très récemment une tentative de généralisation de notre travail a été proposée [Beldiceanu and Petit, 2004]. C'est un sujet qui sera certainement étudié dans le futur. Par ailleurs, on peut imaginer la définition de nombreuses autres contraintes globales molles dérivant des contraintes globales les plus habituelles, ainsi que la description d'algorithmes de filtrage associés.

Chapitre 5

Applications

J'ai travaillé sur de nombreuses applications. Pour certaines j'ai obtenu de très bons résultats puisque j'ai fermé plusieurs problèmes ouverts. Je peux citer les problèmes de "car sequencing", du "all interval series", du "quasigroup completion" ou encore de "sports scheduling".

J'ai décidé de détailler dans cette section deux applications que je considère comme étant majeure : la recherche de la taille de la plus grande clique d'un graphe et le dimensionnement de réseau de télécommunication.

5.1 Recherche de la taille de la plus grande clique d'un graphe

J'ai étudé ce problème [Régin, 2003a, Régin, 2003b] dans le but de promouvoir la PPC, mais aussi dans le but de contredire une idée reçue à propos de la PPC : "la PPC n'est pas une méthode efficace pour résoudre les problèmes purs d'optimisation combinatoire". Par "problèmes purs", on entend habituellement les problèmes qui n'impliquent qu'un seul type de contraintes.

Une **clique** d'un graphe $G = (X, E)$ est un sous-ensemble V de X, tel que toute paire de nœuds de V soit reliée par une arête de E. le problème de la **clique maximum** consiste à trouver $\omega(G)$ la taille de la clique de plus grande cardinalité. trouver une clique de taille k est un problème NP-difficile. Ce problème est important car il apparaît dans de nombreuses applications réelles. Aussi, presque tous les types de méthode ont été utilisés pour le résoudre (voir l'excellent état de l'art de Bomze, Budinich, Pardalos et Pelillo [Bomze et al., 1999]).

Fahle [Fahle, 2002] a été le premier à proposer un modèle de PPC pour résoudre ce problème. L'algorithme de Fahle essaie de construire une clique de plus en plus grande en sélectionnant successivement un nœud et en étudiant **l'ensemble des candidats**, noté *Candidate*, c'est-à-dire l'ensemble des nœuds qui peuvent étendre la clique courante en construction, appelé **ensemble des nœuds courants**, noté *Current*. Après chaque sélection de nœud, des algorithmes de filtrage sont déclenchés afin de supprimer des nœuds de l'ensemble des candidats. L' algorithme que j'ai proposé fonctionne selon le même principe. La difficulté est de définir des algorithmes de filtrage qui soient à la fois efficaces en terme de quantité de nœuds supprimés et aussi en temps. Il existe un algorithme de filtrage de base qui consiste à éliminer les nœuds qui ne sont pas connectés avec le nœud qui vient d'être introduit dans l'ensemble des nœuds courants. Ce filtrage est bien entendu utilisé par l'algorithme de Fahle et le notre. Il est important de noter que cet algorithme à la capacité d'effectuer plus de 6 millions de backtracks par seconde. Aussi, même si cet algorithme s'avère peu efficace en terme de suppressions, il est, en revanche, par-

ticulièrement efficace en temps. L'algorithme de Fahle utilise un second filtrage qui consiste à rechercher un majorant du nombre de couleurs nécessaires pour colorier les voisins d'un nœud. Ce majorant est également un majorant de notre problème. Un nœud sera alors éliminé si ce nombre n'est pas plus grand que la plus grande clique trouvée jusqu'alors.

Nous présentons maintenant notre algorithme. Auparavant, nous rappelons quelques définitions de la théorie des graphes.

Une **ensemble stable** d'un graphe $G = (X, E)$ (independant set en anglais) est un sous-ensemble S de X, tel que toute paire de nœuds de V ne soit pas reliée par une arête de E. Le problème de **l'ensemble indépendant maximum** consiste à trouver $\alpha(G)$ la plus grande cardinalité d'un ensemble indépendant.

Un **ensemble transversal** d'un graphe $G = (X, E)$ (vertex cover en anglais) est un sous-ensemble V de X, tel que toute arête de E ait une extrémité dans V. Le problème de **l'ensemble transversal minimum** consiste à trouver $\nu(G)$ la plus petite cardinalité d'un ensemble transversal.

Une **couplage** d'un graphe $G = (X, E)$ est un sous-ensemble M de E, tel qu'il n'existe pas deux arêtes de M ayant une extrémité commune. Le problème du **couplage maximum** consiste à trouver $\mu(G)$ la plus grande cardinalité d'un couplage.

5.1.1 Un premier filtrage

Etant donné un nœud x nous noterons $\omega(G, x)$ la taille de la plus grande clique contenant x, cela nous permet de définir la propriété à la base du filtrage.

Propriété 6 *Soient $G = (X, E)$ un graphe, x un nœud de G et K un clique de G. Si $\omega(G, x) < |K|$ alors $\omega(G) = \omega(G - \{x\})$.*

Il existe de nombreuses relations connues entre les différents problèmes mentionnés précédemment [Golumbic, 1980] :

• la taille de la plus grande clique est égale à la taille du plus grand ensemble stable dans le graphe complémentaire

• la taille du plus grand ensemble stable est ègale au nombre de sommet du graphe diminué de la taille du plus petit ensemble transversal.

• la taille de la plus grande clique est égale au nombre de sommet du graphe diminué de la taille du plus petit ensemble transversal dans le graphe complémentaire. Formellement, on a :

Propriété 7 *Soit $G = (X, E)$ un graphe, alors*
• $\omega(G) = \alpha(\overline{G})$
• $\alpha(G) = |X| - \nu(G)$
• $\omega(G) = |X| - \nu(\overline{G})$

Propriété 8 *Soit $G = (X, E)$ un graphe, alors*
$\nu(G) \geq \mu(G)$ et on a l'égalité si G est biparti.

On a alors immédiatement :

Propriété 9 $\omega(G) \leq |X| - \mu(\overline{G})$

On peut utiliser cette borne supérieure, mais cela présente un inconvénient : $\overline{G}$ peut ne pas être biparti et l'algorithme pour calculer un couplage maximum pour un graphe non biparti est complexe. C'est pourquoi, nous proposons une autre borne, qui est en fait meilleure, et surtout qui peut s'implémenter de façon beaucoup plus efficace.

Définition 31 *Soit $G = (X, E)$ un graphe. Le* **graphe dupliqué** *de G est le graphe biparti $G^d = (X, Y, F)$, tel que Y soit une copie des nœuds de X, $c(u)$ soit le nœud de Y correspondant au nœud u de X, et il y ait une arête $(u, c(v))$ dans F si et seulement si (u, v) appartient à E.*

Notons que si $(u, v) \in E$ alors $(v, u) \in E$.

Propriété 10 *$\mu(G^d) \geq 2.\mu(G)$ et il existe des graphes G avec $\mu(G^d) > 2.\mu(G)$*

Définissons la projection d'un couplage de G^d dans G :

Définition 32 *Etant donnés $G = (X, E)$ un graphe, M un couplage de G^d et E' le sous-ensemble de E défini par $(u, v) \in E'$ si et seulement si $(u, c(v)) \in M$ ou $(v, c(u)) \in M$. La* **projection** *de M dans G, notée $P(M, G)$, est le sous-graphe de G induit par le sous-ensemble $E' \subseteq E$.*

Propriété 11 *Soient $G = (X, E)$ un graphe, M un couplage de G^d, $P(M, G)$ la projection de M dans G et CC l'ensemble des composantes connexes de $P(M, G)$. Alors*

$$\nu(G) \geq \sum_{cc \in CC} \lceil \frac{|edges(cc)|}{2} \rceil$$

A partir de cette propriété, on peut en déduire une plus simple qui est également plus faible, mais qui est intéressante à cause de la propriété 10.

Propriété 12 $\nu(G) \geq \lceil \frac{\mu(G^d)}{2} \rceil$

On obtient alors à partir de la propriété 7 la propriété recherchée suivante :

Propriété 13 $\omega(G) \leq |X| - \lceil \frac{\mu(\overline{G}^d)}{2} \rceil$

Le premier algorithme de filtrage que nous avons utilisé est basé sur cette propriété appliquée sur le graphe constitué par un nœud donné et ses voisins.

Nous avons choisi cette propriété et non pas la propriété 11 parce qu'avec la propriété 13 on dispose d'un très bon test pour savoir s'il est intéressant de la calculer. En effet il suffit de vérifier si la valeur de $|X| - \lceil \frac{|X|}{2} \rceil$ est plus petite que la plus grande clique trouvée jusqu'alors. D'après nos expérimentations, grâce à ce test seulement 5% des couplages qui sont calculés le sont inutilement. Autrement dit, seuls 5% des nœuds qui passent avec succès le test précédent ne seront pas supprimés par l'algorithme de filtrage.

Nous avons aussi implémenté un algorithme basé sur la propriété 11 mais le gain par rapport à la propriété 13 est réellement faible en terme de nœuds éliminés et plus de temps est requis par l'algorithme. Cela nous amène à un point important de la PPC. La PPC implique un mécanisme de propagation, aussi, et particulièrement pour les problèmes purs, la comparaison de deux problèmes n'est pas simple parce que la propagation doit également être prise en compte. En effet, la combinaison d'une propriété P plus forte qu'une propriété Q avec d'autres propriétés peut conduire avec le mécanisme de propagation au même résultat en remplaçant P par Q. Cet aspect est particulièrement important en PPC, puisqu'il signifie que le meilleur résultat (en temps) n'est pas nécessairement obtenu avec les propriétés les plus fortes. La combinaison des propriétés est en fait, souvent, bien plus importante. C'est exactement le phénomène que l'on observe ici. La propriété 13 et le mécanisme de propagation donne des résultats semblables à ceux obtenus avec la propriété 11 et la propagation. On choisira donc la propriété la plus rapide à calculer. De plus en pratique, il n'est pas nécessaire de calculer systématiquement un couplage maximum. Dès lors qu'un couplage suffisamment grand est trouvé on peut arrêter les calculs, car on sait que l'on ne pourra pas éliminer le nœud considéré.

D'un autre côté, en pratique il y a une différence importante entre la propriété 9 et la propriété 13. Il semble donc que cela vaille réellement la peine d'améliorer la borne supérieure même par une faible valeur, à condition d'avoir un test pertinent pour éviter certains calculs.

5.1.2 Un second filtrage

Pour énumérer l'ensemble des cliques maximales d'un graphe, Bron et Kerbosh [Bron and Kerbosh, 1973] ont proposé d'utiliser un autre ensemble de nœuds : **l'ensemble des nœuds not**, noté Not. Cet ensemble contient les nœuds qui ont déjà été étudiés par l'algorithme, au sens où ils ont déjà été ajoutés à l'ensemble courant précédemment pendant la recherche, et qui sont reliés à l'ensemble des nœuds de l'ensemble $Current$. Nous proposons d'adapter leur idée à notre cas et de la généraliser.

L'idée de Bron et Kerbosh peut être exprimée au travers de la propriété suivante :

Propriété 14 *S'il existe un nœud x dans Not tel que $Candidate \subseteq \Gamma(x)$ alors la branche courante de la recherche peut être abandonnée.*

Cette propriété peut être améliorée lorsqu'on recherche la taille d'une clique maximum. Une propriété de dominance peut alors être établie :

Propriété de dominance 1 *S'il existe un nœud x dans Not tel que $|(Candidate \cup Current) - \Gamma(x)| \leq 1$ alors la branche courante de la recherche peut être abandonnée.*

Dans l'algorithme de Bron et Kerbosh, un nœud est supprimé de Not lorsque le nœud sélectionné n'est pas relié avec lui. Dans notre cas, nous modifions légèrement cette propriété : un nœud est supprimé de Not quand deux nœuds de $Current$ ne sont pas reliés avec lui. La propriété précédente reste valable avec cette nouvelle définition.

Appelons **nœud marqué** un nœud de Not qui n'est pas relié a un nœud de $Current$. A partir de la propriété 1 on peut alors définir un algorithme de filtrage :

Propriété de dominance 2 *Soient y un nœud de $Candidate$ et x un nœud de Not.*
Si $(\Gamma(y) \cap Candidate) \subseteq \Gamma(x)$ alors y peut être supprimé de $Candidate$.
Si x n'est pas marqué et $|(\Gamma(y) \cap Candidate) - \Gamma(x)| \leq 1$ alors y peut être supprimé de $Candidate$.

Malheureusement, il est coûteux de tester cette propriété et en pratique ce coût est prohibitif. Aussi, nous avons préféré l'utiliser de façon différente. Au lieu de chercher si le voisinage de chaque nœud de l'ensemble $Candidate$ est inclus dans le voisinage d'un nœud de Not, nous avons décidé de limiter cette étude aux nœuds dont le voisinage contient tous les nœuds de l'ensemble $Candidat$ sauf, éventuellement, un noeud.

L'application stricte de la propriété de dominance 2 conduit à une réduction du nombre de backtrack d'environ 10%. Cependant, l'implémentation relâchée de cette propriété est plus efficace en terme de temps de calcul, parce que l'on doit uniquement comparer le voisinage d'un nœud de Not avec l'ensemble $Candidate$ et non pas avec chaque élément de cet ensemble pris séparément.

5.1.3 Stratégie de recherche

Comme il a été montré par Bron et Kerbosh pour énumérer les cliques maximales, il est intéressant de sélectionner un nœud tel que la propriété 14 soit satisfaite le plus vite possible.

Cela signifie que lorsqu'un nœud est ajouté à Not, on identifie tout d'abord le nœud x de Not qui a la plus grand nombre de voisins dans l'ensemble des candidats. Ensuite, on sélectionne comme prochain nœud, un nœud y qui n'est pas relié à x.

Nous avons utilisé exactement la même idée en considérant les nœuds non marqués de Not. Lors d'une égalité entre deux nœuds, par rapport au critère considéré, on choisira le nœud y ayant le plus petit nombre de voisins, dans le but d'avoir plus de chance de supprimer rapidement y et donc de satisfaire la propriété de dominance 1.

Lorsqu'un nœud sélectionné ne conduit pas immédiatement à un échec, c'est-à-dire quand la dernière sélection a été un succès, on sélectionne comme prochain nœud celui qui a le plus grand nombre de voisins dans l'ensemble des candidats. Cette approche donne plus de chance pour trouver rapidement des cliques dont la cardinalité est grande. Cet aspect est important puisque les algorithmes de filtrage prennent en compte la taille de la plus grande clique trouvée jusqu'alors.

5.1.4 Technique de plongée

Cette technique ("diving technique" en anglais) est souvent utilisée par les approches MIP. Elle consiste à rechercher de façon incomplète une solution pour chaque valeur de chaque variable. Un algorithme glouton est utilisé pour chaque recherche, donc aucun backtrack n'est autorisé. Ensuite, l'objectif est positionné par rapport à la meilleure valeur trouvée. L'intérêt de cette approche est triple : sont coût est faible puisque la recherche de solution est faite à l'aide d'un algorithme glouton, la valeur minimum de l'objectif peut-être améliorée et on peut trouver une clique dont la taille n'aurait pas facilement été trouvée par une recherche en profondeur d'abord classique. En fait, une recherche systématique perd beaucoup de temps à prouver des optimalités locales, alors que celles-ci peuvent être abandonnées dès qu'une meilleure valeur est trouvée.

Nous avons utilisé cette technique après dix minutes de calcul. Autrement dit, nous arrêtons la recherche classique, nous appliquons cette technique, puis nous continuons la recherche classique en tenant compte de la nouvelle valeur minimale trouvée pour l'objectif et calculée par la technique de plongée.

5.1.5 Résultats expérimentaux

ILOG Solver a été utilisé et nous avons testé notre approche avec l'ensemble de benchmark de DIMACS [Dimacs, 1993].

Toutes les expériences ont été réalisées sur un Pentium IV mobile cadencé à 2Ghz avec 512 Mo de mémoire.

Nous avons limité le temps de résolution de chaque problème à 4 heures (soit 14 400s), excepté pour le problème p_hat1000-2 car après 4 heures nous nous sommes aperçu que le problème pouvait être résolu (en fait il a fallu 16 845s). Les résultats sont détaillés dans [Régin, 2003a].

Voici les résultats donnés par notre méthode :

- Tous les problèmes ayant 400 nœuds ou moins sont résolus. Pour la première fois la série des brock400 est résolue. Seul le problème johnson32-2-4 nous empêche de résoudre tous les problèmes ayant 500 nœuds.
- Tous les problèmes de la série san sont résolus.
- 7 problèmes ont été fermés pour la première fois : toute la série des brock400 , p_hat500-3, p_hat1000-2 et sanr200_0.9, qui est résolu en 150 s.
- deux résultats sont particulièrement remarquables : p_hat300-3 est résolu en 40s et p_hat700-2 en 255s. Soit au moins 10 fois plus vite qu'avec n'importe quelle autre méthode.

– Les bornes inférieures de deux problèmes restant ouverts ont été améliorées : MANN_a45 (la valeur optimale est atteinte) et MANN_a81.
– Pour 6 problèmes les meilleures bornes inférieures connues sont atteintes : p_hat700-3, johnson32-2-4, hamming10-4, keller5 (la valeur optimale est atteinte), MANN_a45 (la valeur optimale est atteinte) et MANN_a81.

Certains auteurs ont trouvé de meilleures bornes inférieures que nous. C'est le cas de [Balas and Niehaus, 1996] pour p_hat1500-2 (65) et p_hat1500-3 (94) ; et de [Homer and Peinado, 1996] pour keller6 (59) and p_hat1000-3 (68).

Nous pensons que notre méthode n'est pas la bonne pour résoudre certains problèmes : keller6, johnson32-2-4, hamming10-4. Pour les autres problèmes ouverts nous sommes plus confiant.

Comparaison avec des méthodes complètes

Nous avons comparé notre approche avec 3 autres algorithmes :
• [Wood, 1997] : Un algorithme de branch-and-bound est associé avec un calcul de coloriage fractionnel.
• [Östegard, 2003] : cette approche est semblable à la programmation dynamique : on résout la problème avec un nœud, puis avec 2 nœuds et ainsi de suite jusqu'à n. Pour chaque résolution les valeurs optimales précédemment calculées sont utilisées pour le nouveau problème considéré.
• [Fahle, 2002] : Fahle utilise comme nous une méthode de PPC en considérant deux filtrages : le premier supprime les nœuds qui ont un degré trop faible pour améliorer la valeur courante de l'objectif ; le second calcule pour chaque nœud un majorant du nombre de couleurs nécessaires pour colorier le graphe induit par les voisins de ce nœud. La stratégie sélectionne le nœud ayant le plus petit degré.

Le tableau suivant résume cette comparaison :

	Wood	Östegard	Fahle	Régin
nombre problèmes résolus	38	36	45	52
nombre problèmes résolus en moins de 10 min.	38	35	38	44
nombre de meilleurs temps	15	26	10	30
nombre de meilleures bornes inf. pour des problèmes ouverts	0	0	1	5

La méthode proposée par Östegard est réputée pour sa rapidité de résolution de certains problèmes. Si nous considérons l'ensemble des problèmes résolus par Östegard en moins de 10 minutes alors Östegard a besoin de 345s pour résoudre tous ces problèmes alors que notre méthode n'a besoin que de 282s.

Notre approche est presque toujours meilleure que celle de Fahle, seul p_hat1500-1 est résolu plus vite par Fahle.

Comparaison avec des méthodes incomplètes

Deux méthodes heuristiques donnent de très bons résultats pour résoudre le problème de la clique maximum : QUALEX-MS [Busygin, 2002] et la méthode proposée par [St-Louis et al., 2003].

Pour l'ensemble de tests que nous avons considéré, ces deux méthodes atteignent les meilleures bornes inférieures calculées pour 50 problèmes. En moins de 1 minute, QUALEX en trouve 48.

En comparaison, voici un récapitulatif des résultats obtenus par notre méthode :

– Avec une limite de 4 heures par problème, notre méthode atteint la meilleure borne inférieure connue pour 58 problèmes, et pour 52 de ces problèmes l'optimalité est prouvée.
– Avec une limite de 10 min par problème, 49 meilleures bornes inférieures sont atteintes et 44 sont prouvées optimales.
– Avec une limite de 1 min, 41 meilleures bornes inférieures sont atteintes et 37 sont prouvées optimales.

Ces résultats montrent que notre méthode est compétitive par rapport aux meilleures méthodes incomplètes, même si le temps de calcul est limité.

5.1.6 Perspectives

La version du problème que nous avons considéré n'implique pas de coûts sur les arcs. Il serait certainement intéresser de tester notre approche avec le problème de la recherche de la clique de poids maximum. Bien entendu, les filtrages devront être modifiés, mais nous ne perdons pas espoir quant à la possibilité d'introduire de nouveaux algorithmes de filtrage capable de prendre en compte efficacement des coûts sur les arcs.

5.2 Dimensionnement de réseau de télécommunication

Ce problème a été étudié dans le cadre du projet RNRT ROCOCO. La présentation qui suit est inspirée de [Bernhard et al., 2002]. On trouvera plus de détails dans [Le Pape et al., 2002]. Le modèle de PPC pour résoudre ce problème a été proposé par C. Le Pape et moi-même.

Les moyennes et grandes entreprises échangent de plus en plus de données sur les réseaux connectant leurs sites. Par conséquent l'achat ou l'évolution d'un réseau est devenu un élément stratégique pour ces entreprises. Elles y consacrent une part toujours plus importante de leur budget de fonctionnement et ceci les obligent à être de plus en plus exigeantes vis à vis des opérateurs de réseau : lors de l'achat ou d'évolutions de leur réseau, elles font jouer fortement la concurrence et imposent aux opérateurs des contraintes qui ne peuvent pas toujours être anticipées par ces derniers.

Cette section présente le prototype de PPC qui a été élaboré.

5.2.1 Description du problème

Soit $G = (X, U)$ un graphe orienté. A ce graphe est associé un ensemble de d demandes. Chaque demande associe à un couple de sommets (p, q) une valeur entière Dem_{pq} représentant une quantité de flot à transporter de p vers q. Nous utiliserons un triplet (p, q, Dem_{pq}) pour représenter une telle demande. Un unique chemin de p vers q doit être choisi pour router cette demande. Autrement dit, chaque demande doit être satisfaite en utilisant un et un seul chemin de transport. C'est ce qu'on appelle le problème du monoroutage.

Une fonction $Bmax$ associe à chaque demande (p, q, Dem_{pq}) une limite $Bmax_{pq}$ sur le nombre de bonds, i.e., sur le nombre d'arcs utilisés pour router la demande. En particulier, si $Bmax_{pq} = 1$, la demande de p vers q doit être routée directement sur l'arc (p, q).

Une fonction $Tmax$ associe à chaque nœud i une limite $Tmax_i$ sur la quantité de flux traitée par i. Ceci inclut :

- Le trafic au départ de i, i.e.,$\sum_{q \neq i} Dem_{iq}$.
- Le trafic arrivant à i, i.e., $\sum_{p \neq i} Dem_{pi}$.

• Le trafic transitant par i, c'est-à-dire la somme des demandes Dem_{pq}, $p \neq i$, $q \neq i$, pour lesquelles le chemin choisi passe par i.

Notons que le trafic transitant par le nœud i n'est compté qu'une fois. La limite $Tmax_i$ ne s'applique donc pas à la somme des flux entrant et sortant de i. Par contre, il est possible de se ramener à une contrainte sur le flux entrant dans i (qui doit être limité à $Tmax_i - \sum_{q \neq i} Dem_{iq}$) ou, de manière équivalente, à une contrainte sur le flux sortant de i (qui doit être limité à $Tmax_i - \sum_{p \neq i} Dem_{pi}$).

Deux fonctions Pin et $Pout$ associent à chaque nœud i un nombre maximal de ports d'entrée (Pin_i) et de sortie ($Pout_i$).

Pour chaque arc (i, j), K_{ij} capacités possibles $Capa_{ij}^k, 1 \leq k \leq K_{ij}$, sont données. Nous posons par ailleurs $Capa_{ij}^0 = 0$. Une et une seule de ces $K_{ij}+1$ capacités doit être choisie. Cependant, il peut être autorisé de multiplier cette capacité par un entier compris entre une valeur minimale $Wmin_{ij}^k$ et une valeur maximale $Wmax_{ij}^k$ données. Le problème de dimensionnement consiste donc non seulement à choisir pour chaque arc (i, j) une capacité $Capa_{ij}^k$, mais aussi à choisir le coefficient multiplicateur w_{ij}^k retenu. Notons que la capacité $Capa_{ij}^k$ doit nécessairement être retenue dès lors que $Wmin_{ij}^k$ est supérieure ou égale à 1. Il ne peut donc exister qu'un seul k avec $Wmin_{ij}^k > 0$, sinon le problème n'admet pas de solution.

Les choix effectués pour les arcs (i, j) et (j, i) sont liés de la manière suivante : si la k-ième capacité $Capa_{ij}^k$ est retenue avec un coefficient multiplicateur $w_{ij}^k > 0$ pour l'arc (i, j), alors la k-ième capacité $Capa_{ji}^k$ doit être retenue pour l'arc (j, i), avec le même coefficient multiplicateur $w_{ji}^k = w_{ij}^k$.

Une fonction $Cost$ associe à chaque niveau de capacité possible d'un arc (i, j) et de son arc inverse (j, i) une valeur entière $Cost_{ij}^k$, qui représente le coût de la capacité. Ce coût n'est à comptabiliser qu'une fois pour les deux arcs (i, j) et (j, i). Cependant, lorsqu'un coefficient multiplicateur est associé à une capacité, le même coefficient multiplicateur est appliqué au coût.

Une fonction Sec à valeurs booléennes détermine pour chaque demande si le routage de cette demande doit être sécurisé. $Sec_{pq} = 1$ signifie que le routage doit être sécurisé. $Sec_{pq} = 0$ signifie que la sécurisation n'est pas nécessaire. Lorsque le routage doit être sécurisé, il est interdit de passer par un nœud ou par un arc non sécurisé. Pour chaque nœud i, un indicateur de risque $Risk_i$ indique si le nœud est sécurisé ou non. De même, pour chaque arc (i, j) et chaque $k, 1 \leq k \leq K_{ij}$, un indicateur de risque $Risk_{ij}^k$ indique si l'arc dans la configuration k est lui aussi sécurisé. Autrement dit, si $Sec_{pq} = 1$, il est interdit de passer par un nœud intermédiaire tel que $Risk_i = 1$ et il est interdit de passer par un lien tel que $Risk_{ij}^k = 1$.

Le problème consiste à déterminer le chemin à associer à chaque demande et les capacités à affecter aux arcs de façon à satisfaire toutes les demandes, à assurer que pour tout arc la capacité choisie (pondérée par le coefficient multiplicateur retenu) est supérieure à la somme des quantités transportées par l'arc, et enfin à minimiser la somme des cots.

Le problème du dimensionnement de réseau avec mono-routage a été peu étudié. Rothlauf, Goldberg et Heinzl [Rothlauf et al., 2002] ont travaillé sur des problèmes similaires à 15, 16 et 26 nœuds, fournis par Deutsche Telekom, mais en obligeant le réseau solution à être un arbre. Gabrel, Knippel et Minoux [Gabrel et al., 1999] ont développé une méthode exacte pour résoudre des problèmes de dimensionnement de réseaux de 8 à 20 nœuds avec fonctions de coût en escalier, mais en considérant un routage de type multiflot. De fait, les problèmes avec multiflot ont été les plus étudiés, avec divers types de fonctions de coût, par exemple un coût fixe et un coût proportionnel au trafic comme dans [Gendron and Crainic, 1994].

5.2.2 Résolution du problème en PPC

Une utilisation naive d'un outil de programmation par contraintes comme ILOG Solver ne permet pas de résoudre le problème dans des conditions d'efficacité satisfaisantes : les premières expériences menées par France Télécom R&D ont montré qu'à partir de 8 nœuds, il pouvait être difficile de générer des solutions à moins de 20% de l'optimum en un temps de calcul limité à quelques minutes. Il nous a donc paru nécessaire de tirer plus amplement parti de la structure du problème, et notamment du fait qu'il s'agit, pour l'essentiel, de choisir des chemins pour router des communications dans un graphe.

J'ai introduit un nouveau type de variable représentant un chemin d'un nœud source donné vers un nœud puits donné. Plus précisément, un chemin est défini par deux variables ensemblistes, représentant l'ensemble des nœuds et l'ensemble des arcs du chemin, et un ensemble de contraintes entre ces deux variables ensemblistes :

- Si un arc appartient au chemin, ses deux extrémités appartiennent au chemin.
- Un et un seul arc sortant de la source du chemin doit appartenir au chemin.
- Un et un seul arc entrant dans le puits du chemin doit appartenir au chemin.
- Si un nœud différent de la source et du puits appartient au chemin, alors un et un seul arc entrant dans ce nœud et un et un seul arc sortant de ce nœud doivent appartenir au chemin.

Plusieurs contraintes globales ont par ailleurs été implémentées de manière à détecter les nœuds et les arcs devant appartenir à un chemin donné (pour des raisons de connexité), éliminer les nœuds et les arcs ne pouvant pas appartenir à un chemin donné, raisonner sur le nombre de nœuds et d'arcs d'un chemin, et relier les "variables chemins" ainsi définies aux variables définissant la capacité et le niveau de sécurité (risqué ou non) des nœuds et des arcs du réseau.

La plus importante de ces contraintes globales identifie les nœuds et les arcs par lesquels un chemin doit impérativement passer, en tenant compte des informations disponibles sur le nombre maximal $Bmax$ d'arcs du chemin. L'algorithme utilisé pour propager cette contrainte est le suivant :

1. Utilisation de l'algorithme de Ford (cf. [Gondran and Minoux, 1985]) pour identifier le plus court chemin admissible (en nombre d'arcs) entre la source et chaque nœud du graphe ;
2. Utilisation de l'algorithme de Ford pour identifier le plus court chemin admissible (en nombre d'arcs) entre chaque nœud du graphe et le puits ;
3. Utilisation des longueurs de chemins ainsi calculées pour éliminer les nœuds par lesquels ne peut passer aucun chemin de longueur inférieure ou égale à Bmax arcs ;
4. Utilisation des longueurs de chemins ainsi calculées pour marquer les nœuds qui peuvent trivialement être contournés par un chemin de longueur inférieure ou égale à $Bmax$ arcs ;
5. Utilisation pour chaque nœud non marqué de l'algorithme de Ford dans le but de déterminer s'il existe un chemin de la source au puits de longueur inférieure ou égale à $Bmax$ arcs ne passant pas par le nœud considéré.

L'utilisation de cet algorithme s'est avérée globalement rentable, malgré sa complexité en $O(nmBmax)$, soit $O(n^4)$ dans le pire des cas.

Nous avons par ailleurs utilisé un algorithme de plus court chemin en tant qu'heuristique pour guider la recherche de bonnes solutions. La stratégie utilisée est extrêmement simple. A chaque étape de la résolution, on choisit le chemin non instancié pour lequel l'expression $2Dem_{pq} + Dem_{qp}$ est la plus forte. Ceci permet de pondérer l'importance donnée à une demande de p vers q par le poids de la demande inverse qui, même lorsque le paramètre $symdem$ n'est pas positionné, a dans la solution optimale de fortes chances de suivre un chemin de q vers p symétrique

de celui suivi de p de q. Nous déterminons alors le chemin le moins coûteux (marginalement compte-tenu des demandes déjà routées) pour router cette demande. Un point de choix est alors créé : dans la première branche, la demande est contrainte à passer par le dernier arc de ce chemin qui n'est pas encore imposé ; en cas de backtrack, cet arc est au contraire interdit pour cette demande. Lorsqu'un chemin est totalement instancié, nous passons à la demande suivante. Une nouvelle solution est obtenue lorsque tous les chemins sont instanciés. La recherche continue alors en contraignant le coût des nouvelles solutions à être strictement inférieur au coût de la meilleure solution déjà trouvée. Pour favoriser l'obtention rapide de bonnes solutions, nous avons encapsulé cette procédure de résolution dans un mécanisme de Slice-Based Search (SBS), une implantation du Discrepancy Bounded Depth First Search [Beck and Perron, 2000] dans lequel un maximum de recalculs sont évités.

5.2.3 Résultats expérimentaux

Nous avons considéré 3 séries de problèmes : A, B et C. Pour chaque série, 6 paramètres booléens sont utilisés :

- *sec* indique que les contraintes de sécurité doivent être prises en compte.
- *nomult* interdit l'utilisation de multiplicateurs de capacités.
- *symdem* impose la propriété de symétrie des routages.
- *bmax* indique que les contraintes de nombre de bonds pour chaque demande doivent être prises en compte.
- *pmax* indique que les contraintes de nombre de ports en chaque nœud doivent être prises en compte.
- *tmax* indique que les contraintes de trafic maximal à chaque nœud doivent être prises en compte.

Les paramètres *sec*, *pmax* et *tmax* sont ignorés par les séries B et C. On obtient donc $2^6 = 24$ variantes pour la série A et 8 variantes pour les séries B et C. Toutes les variantes doivent être résolues dans un temps fixe de 10 minutes.

Le tableau suivant résume les résultats obtenus. Le nombre suivant la série indique le nombre de nœuds du réseau. Pour chaque instance, nous sommons les valeurs obtenues pour chaque paramètrage après un maximum de 10 minutes de temps de calcul par problème sur un PC Pentium III à 700 MHz. Le tableau fournit pour chaque instance, la somme des meilleures bornes inférieures connues (lorsque ce nombre est significatif), la somme des coûts des meilleures solutions connues, la somme des coûts des solutions trouvées par notre algorithme (PPC) et la somme des coûts des solutions trouvées par deux autres algorithmes à base de programmation linéaire en nombres entiers (une variante du modèle MIP précédent) et de génération de colonnes (GC), toujours avec une limite de 10 minutes de temps de calcul. Un fail dans le tableau signifie qu'il existe au moins un paramètrage pour lequel l'algorithme ne donne aucune solution en 10 minutes. Le tableau fournit également pour chaque algorithme l'écart relatif moyen aux meilleures solutions connues. Notons que quand la taille du problème augmente, le MIP ne parvient pas toujours à générer au moins une solution entière.

L'algorithme de PPC est le plus robuste. C'est également celui qui est considéré comme le plus intéressant par France Telecom R&D (voir le rapport final du projet ROCOCO). Cependant, sur les séries B et C, les résultats ne sont pas toujours satisfaisants en terme d'écart. En particulier, sur B11, B12 et C12, la PPC ne fournit pas les meilleurs résultats en moyenne.

#pb	$\sum$ Binf	$\sum$ Bsup	$\sum$ PPC	Ecart PPC	$\sum$ MIP	Ecart MIP	$\sum$ GC	Ecart GC
A04	1782558	1782558	1782558	0.00%	1782558	0.00%	1782558	0.00%
A05	2351778	2351778	2351778	0.00%	2351778	0.00%	2351778	0.00%
A06	2708264	2708264	2708264	0.00%	2708264	0.00%	2708264	0.00%
A07	3290590	3290590	3290940	0.01%	3337284	1.42%	3310007	0.60%
A08	4002083	4050151	4076785	0.69%	4417611	9.06%	4263830	5.11%
A09	4374737	4966858	5027246	1.25%	5934187	19.44%	5621264	12.85%
A10	4958292	5843478	5934297	1.57%	fail	fail	fail	fail
B10		155748	172255	10.62%	174494	12.04%	176625	13.40%
B11		271080	320356	19.20%	302226	12.46%	300317	11.70%
B12		207424	235412	13.49%	235031	13.32%	227387	9.62%
C10		102824	104686	1.84%	106183	3.24%	105578	2.72%
C11		169124	179078	5.90%	184485	9.11%	199265	17.83%
C12		310788	360673	16.20%	fail	fail	348666	12.26%

Une étude des paramètrages posant le plus de difficulté à chacun de ces algorithmes montre que comparativement :

• les contraintes qui posent le plus de difficultés à l'algorithme de génération de colonnes sont les contraintes de sécurité (sec), de nombre de ports maximal ($pmax$) et de trafic maximal ($tmax$) ; par contre, la présence de la contrainte de nombre de bonds ($bmax$) est un élément favorable ;

• la contrainte qui pose le plus de difficultés à l'algorithme de programmation par contraintes est la contrainte de trafic maximal ;

• les contraintes qui posent le plus de difficultés au MIP sont les contraintes de nombre de bonds et de trafic maximal ;

• l'algorithme de programmation par contraintes semble tirer comparativement moins bien parti que les deux autres de la contrainte de chemins symétriques ($symdem$), alors que dans les trois cas le nombre de chemins est divisé par deux.

5.2.4 Perspectives

La définition d'une variable correspondant à un graphe est une idée qui mérite d'être développée, tout comme l'idée d'instancier directement des chemins et non pas tous les arcs individuellement en essayant de garantir la propriété de chemin.

Chapitre 6

Réflexions et perspectives

Nous présentons dans cette section tout d'abord quelques remarques d'ordre général à propos des algorithmes de filtrage, puis nous mentionnons quelques idées concernant les méthodes constructives qui s'opposent un peu aux méthodes de filtrage.

6.1 Qualité d'un algorithme de filtrage

Cette section est inspirée d'une section semblable de [Régin, 2003]. Nous allons essayer d'identifier les propriétés que doit satisfaire un bon algorithme de filtrage.

En section 3.1 nous avons présenté un algorithme de filtrage générique permettant de calculer la fermeture par consistance d'arc à partir d'une fonction capable de déterminer si un problème P admet une solution. La complexité de cet algorithme est $nd \times O(P)$, où $O(P)$ est la complexité du test de consistance du problème P. Il est donc inutile de développer un algorithme dédié qui conduirait à la même complexité.

A partir de cette remarque j'ai proposé la classification suivante :

Définition 33 *Soit C une contrainte dont la consistance peut être calculée en $O(C)$. Un algorithme de filtrage réalisant la consistance d'arc pour C est*

- **pauvre** *si sa complexité est en $O(nd) \times O(C)$;*
- **moyen** *si sa complexité est en $O(n) \times O(C)$;*
- **bon** *si sa complexité est en $O(C)$;*

Plusieurs bons algorithmes de filtrage sont connus pour certaines contraintes. C'est par exemple le cas pour la contrainte alldiff, ou la contrainte globale de cardinalité .

Plusieurs algorithmes de filtrage moyens ont été développés pour certaines contraintes. C'est le cas pour la contrainte symmetric alldiff ou la contrainte globale de cardinalité avec coûts.

Cependant les algorithmes bons ne sont pas parfaits. En effet la classification proposée est basée sur la complexité dans le pire des cas. L'aspect incrémental des algorithmes est particulièrement important en PPC puisque les algorithmes vont être appelés de très nombreuses fois, c'est pourquoi nous proposons une nouvelle catégorie :

Définition 34 *Un algorithme de filtrage réalisant la consistance d'arc est* **parfait** *s'il a toujours le même coût que le test de consistance de la contrainte.*

Cette définition signifie que la complexité doit être la même dans tous les cas et pas seulement pour le pire des cas. En fait, cela signifie que le filtrage ne coûte rien

par rapport au test de consistance de la contrainte. Un tel algorithme de filtrage est rare. Par exemple, il n'en existe pas pour la contrainte alldiff, parce la consistance peut parfois être testée en $O(1)$ (une arête a été supprimée et elle n'appartient pas au couplage maximum) alors que le filtrage pourra nécessiter de balayer toutes les arêtes du graphe des valeurs.

La seule contrainte que nous ayons trouvée pour laquelle un algorithme de filtrage parfait est connu est la contrainte $(x < y)$.

Deux autres points jouent un rôle important dans la qualité d'un algorithme de filtrage : l'incrémentalité et la complexité amortie.

Comme nous l'avons dit, l'aspect incrémental des algorithmes est particulièrement important en PPC puisque les algorithmes vont être systématiquement appelés après chaque modification du domaine d'une variable impliquée dans la contrainte. Néanmoins, un algorithme de filtrage ne doit pas uniquement se focaliser sur ce point. En effet, il est parfois plus rapide de recommencer les calculs à partir de rien. Autrement dit, l'incrémentalité a aussi ses limites. Cela est clairement montré dans le cas de contraintes binaires données en extension [Bessière and Régin, 2001, Régin, 2004].

Il y a plusieurs manières d'améliorer le comportement incrémental d'un algorithme :

- Les calculs précédents sont pris en compte lorsqu'un nouveau calcul doit être effectué. C'est par exemple l'idée de la dernière valeur étudiée pour les algorithmes de type AC-7.
- L'algorithme de filtrage n'est pas appelé systématiquement après chaque modification. C'est le cas des contrainte "pushées" de ILOG Solver par exemple.
- Des conditions suffisantes pour qu'il n'y ait aucune suppression sont identifiées et l'algorithme de filtrage n'est effectivement appelé que si aucune de ces conditions n'est satisfaite.

Par ailleurs, lorsqu'un algorithme de filtrage est incrémental on peut espérer être capable de calculer sa complexité amortie par suppression ou pour une branche de l'arbre de recherche, ce qui est plus réaliste.

6.2 Approche constructive

La PPC est fortement basée sur l'utilisation d'algorithme de filtrage. On est alors en droit de se poser une question fondamentale : Pourquoi élimine t'on les valeurs des domaines des variables ?

On peut répondre à cette question en disant que c'est nécessaire pour le mécanisme de propagation ou pour la communication entre les contraintes. Cependant, même si ces réponses sont acceptables, elles ne donnent pas de réponse fondamentale.

Je me suis posé cette question lorsque j'ai terminé mon Doctorat. La réponse suivante m'est alors venue à l'esprit : on élimine des valeurs afin de limiter le nombre de mauvais choix réalisés par la procédure de recherche.

On peut alors voir la PPC sous un angle différent : à partir de domaines initiaux de valeurs on va éliminer celles qui ne peuvent pas appartenir à une solution afin d'éviter de les choisir ensuite. J'ai alors eu l'idée d'une nouvelle approche : au lieu d'avoir une démarche destructrice, autrement dit procédant par suppressions de valeurs, ne peut-on pas avoir une démarche constructive ? Cette approche constructive procède de façon complètement différente de l'approche classique. On commence par considérer des domaines vides, puis on va essayer des rajouter des valeurs dans ces domaines de telle façon que l'ensemble des domaines ainsi construit soit cohérent,

c'est-à-dire que chaque valeur de chaque domaine appartienne à une solution du problème en considérant ces domaines construits. Cette idée va bien entendu être combinée avec le mécanisme de filtrage.

Ce principe s'avère particulièrement utile lorsque les domaines initiaux sont très grand et lorsque les contraintes entre les variables admettent peu de solutions. Cela signifie que tout choix va avoir d'énormes conséquences. Dans ce cas, il est légitime de se demander pourquoi on s'évertue à vérifier la consistance d'un grand nombre de valeurs avec les contraintes alors que dès que l'on va faire un choix de très nombreuses valeurs vont immédiatement disparaître. On perd donc beaucoup de temps. Ce type de problème se rencontre très fréquemment lorsque l'on étudie des problèmes de configuration. En effet pour ces problèmes les domaines sont bien souvent définis à partir de base de données et les valeurs sont peu compatibles entre elles. Par exemple les valeurs vont correspondre à des boulons et des écrous et les contraintes exprimeront les compatibilités entre ces boulons et ces écrous. Certaines idées du moteur de résolution ILOG Configurator sont proches de la méthode constructive que nous venons de mentionner.

J'ai proposé avec T. Schiex, C. Gaspin et G. Verfaillie un algorithme, appelé Lazy-AC, basé sur cette idée [Schiex et al., 1996].

Cet article essaie aussi de montrer comment cette approche peut-être adaptée afin de pouvoir bénéficier des avancées apportées par de nombreux travaux en PPC. Par exemple, comment peut-on utiliser les stratégies de recherche habituelles avec une approche constructive, en l'occurrence choisir la variable dont le domaine est le plus petit ?

En fait cette approche est séduisante, mais elle pose de nombreux problèmes en pratique. Notamment, parce qu'elle demande le développement d'un mécanisme de génération de valeurs qui soit exact et performant. Or, à ce jour, un tel mécanisme ne semble pas exister. Le problème majeur est que l'implémentation incrémentale des filtrages s'avère relativement efficace en pratique, peut-être parce qu'elle a tout simplement été plus étudiée. Les notions d'incrémentalité et de retour-arrière, c'est-à-dire la restauration de l'état précédent ou d'un état équivalent, n'ont pas encore, à l'heure actuelle trouvé de solutions satisfaisantes. Un autre problème est que certaines modélisations avec la PPC classique, par exemple les domaines hiérarchiques, permettent de résoudre certains défauts d'un modèle plus naif. Le lecteur plus particulièrement intéressé pourra consulter les travaux réalisés dans le cadre des problèmes de configuration.

L'approche constructive reste néanmoins une voie de recherche qui mériterait certainement d'être plus étudié.

Chapitre 7

Conclusion

Au travers de ce document j'ai essayé de montrer ma contribution personnelle au domaine, à savoir :

- divers principes généraux de modélisation ;
- des algorithmes génériques de filtrage pour les contraintes binaires ou non
- de nombreuses contraintes globales fondamentales associées à des algorithmes de filtrage originaux ou intégrant des algorithmes de RO. Le filtrage réalisé par ces algorithmes étant le plus souvent caractérisé.
- un nouveau point de vue et un nouveau modèle pour la résolution des problèmes sur-contraints, accompagnés de plusieurs algorithmes de filtrage améliorant l'existant ;
- la proposition de nouveaux thèmes comme la définition générale du coût de violation de contraintes ou les contraintes globales molles.
- la résolution de certaines applications particulièrement difficiles

J'ai toujours eu comme souci de proposer des solutions générales pouvant s'intégrer dans un moteur de PPC. J'espère que ce document montre cela.

A la vue des travaux que j'ai réalisés, il pourrait être intéressant d'étudier les voies de recherche suivantes :

- le calcul de l'ensemble des solutions d'un problème afin de pouvoir engendrer des contraintes dont les combinaisons autorisées sont données en extension.
- la combinaison de certaines contraintes globales avec d'autres contraintes simples afin d'obtenir de nouvelles contraintes globales plus fortes encore.
- l'exploitation plus fine de certains algorithmes de RO ou de théorie des graphes ou la définition de nouveaux algorithmes pour réduire la complexité de certains algorithmes de filtrage comme celui réalisant la consistance d'arc pour la contrainte symmetric alldiff, ou pour généraliser d'autres algorithmes de filtrage comme celui de la contrainte globale de cardinalité avec ou sans coûts.
- la prise en compte de disjunctions à la place de la notion de distance dans la contrainte globale de distance minimum afin de pouvoir s'attaquer aux problèmes d'ordonnancement avec la granularité des valeurs des domaines et non pas seulement des bornes.
- la définition de contraintes basée sur des problèmes NP-Complets et l'utilisation d'algorithmes d'approximation pour estimer la consistance de ces contraintes et proposer des algorithmes de filtrage associés.
- l'étude de nouvelles contraintes globales molles et la définition de nouveaux coûts de violation généraux.

Le succès de nombreuses méthodes vient de leur capacité à très bien résoudre au moins un problème particulier. Comme la PPC est une technique à vocation très générale qui n'a pas été créée dans le but de résoudre un problème précis, on ne dispose pas d'une telle information à l'heure actuelle. Plus précisément un tel

problème n'a pas encore été exhibé. Je pense qu'il faudrait essayer d'en trouver un afin de promouvoir la PPC dans les autres communautés.

Bibliographie

[Ahuja et al., 1993] Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows*. Prentice Hall.

[Balas and Niehaus, 1996] Balas, E. and Niehaus, W. (1996). Finding large cliques in arbitrary graphs by bipartite matching. In Johnson, D. and Trick, M., editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 29–52. American Mathematical Society.

[Baptiste et al., 1998] Baptiste, P., Le Pape, C., and Peridy, L. (1998). Global constraints for partial csps : A case-study of resource and due date constraints. In *Proceedings CP'98*, pages 87–101, Pisa, Italy.

[Beck and Perron, 2000] Beck, J.-C. and Perron, L. (2000). Discrepancy-bounded depth first search. In *CP-AI-OR'00*.

[Beldiceanu, 2001a] Beldiceanu, N. (2001a). Pruning for the minimum constraint family and for the number of distinct values constraint family. In *Proceedings CP'01*, pages 211–224, Pathos, Cyprus.

[Beldiceanu, 2001b] Beldiceanu, N. (2001b). Pruning for the minimum constraint family and for the number of distinct values constraint family. *Proceedings CP*, pages 211–224.

[Beldiceanu and Carlsson, 2002] Beldiceanu, N. and Carlsson, M. (2002). A new multi-resource cumulatives constraint with negative heights. In *Proceedings CP'02*, pages 63–79, Ithaca, NY, USA.

[Beldiceanu and Contejean, 1994] Beldiceanu, N. and Contejean, E. (1994). Introducing global constraints in chip. *Journal of Mathematical and Computer Modelling*, 20(12) :97–123.

[Beldiceanu et al., 2001] Beldiceanu, N., Guo, Q., and Thiel, S. (2001). Non-overlapping constraints between convex polytopes. In *Proceedings CP'01*, pages 392–407, Pathos, Cyprus.

[Beldiceanu and Petit, 2004] Beldiceanu, N. and Petit, T. (2004). Cost evaluation of soft global constraints. In *Proceedings CP-AI-OR'04*, pages 80–95, Nice, France.

[Benferhat et al., 1993] Benferhat, S., Cayrol, C., Dubois, D., Lang, J., and Prade, H. (1993). Inconsistency management and prioritized syntax-based entailment. *Proceedings IJCAI*, pages 640–645.

[Berge, 1970] Berge, C. (1970). *Graphe et Hypergraphes*. Dunod, Paris.

[Bernhard et al., 2002] Bernhard, R., Chambon, J., Pape, C. L., Perron, L., and Régin, J.-C. (2002). Résolution d'un problème de conception de réseau avec parallel solver. In *JFPLC*, pages 151–166, Nice, France.

[Bessière and Cordier, 1993] Bessière, C. and Cordier, M. (1993). Arc-consistency and arc-consistency again. In *AAAI-93, Proceedings Eleventh National Conference on Artificial Intelligence*, pages 108–113, Washington, DC.

[Bessière et al., 1999] Bessière, C., Freuder, E., and Régin, J.-C. (1999). Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107(1) :125–148.

[Bessière and Régin, 1994] Bessière, C. and Régin, J.-C. (1994). An arc-consistency algorithm optimal in the number of constraint checks. In *TAI'94, Proceedings IEEE Conference on Tools with Artificial Intelligence*, pages 397–403, New Orleans.

[Bessière and Régin, 1996] Bessière, C. and Régin, J.-C. (1996). Mac and combined heuristics : Two reasons to forsake fc (and cbj ?) on hard problems. In *CP96, Second International Conference on Principles and Practice of Constraint Programming*, pages 61–75, Cambridge, MA, USA.

[Bessière and Régin, 1997] Bessière, C. and Régin, J.-C. (1997). Arc consistency for general constraint networks : preliminary results. In *Proceedings of IJCAI'97*, pages 398–404, Nagoya.

[Bessière and Régin, 1999] Bessière, C. and Régin, J.-C. (1999). Enforcing arc consistency on global constraints by solving subproblems on the fly. In *Proceedings of CP'99, Fifth International Conference on Principles and Practice of Constraint Programming*, pages 103–117, Alexandria, VA, USA.

[Bessière and Régin, 2001] Bessière, C. and Régin, J.-C. (2001). Refining the basic constraint propagation algorithm. In *Proceedings of IJCAI'01*, pages 309–315, Seattle, WA, USA.

[Bistarelli et al., 2002] Bistarelli, S., Codognet, P., and Rossi, F. (2002). Abstracting soft contraint framework. *to appear in Artificial Intelligence*.

[Bistarelli et al., 1995] Bistarelli, S., Montanari, U., and Rossi, F. (1995). Constraint solving over semirings. *Proceedings IJCAI*.

[Bistarelli et al., 1997] Bistarelli, S., Montanari, U., and Rossi, F. (1997). Semiring-based constraint logic programming. *IJCAI*.

[Bistarelli et al., 1999] Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., and Fargier, H. (1999). Semiring-based csps and valued csps : Frameworks, properties, and comparison. *Constraints*, 4 :199–240.

[Blazewicz et al., 1993] Blazewicz, J., Ecker, K., Schmidt, G., and Weglarz, J. (1993). *Scheduling in Computer and Manufacturing Systems*. Springer Verlag.

[Bleuzen-Guernalec and Colmerauer, 1997] Bleuzen-Guernalec, N. and Colmerauer, A. (1997). Narrowing a 2n-block of sortings in o(nlog(n)). In *Proceedings of CP'97*, pages 2–16, Linz, Austria.

[Bomze et al., 1999] Bomze, I., Budinich, M., Pardalos, P., and Pelillo, M. (1999). The maximum clique problem. *Handbook of Combinatorial Optimization*, 4.

[Bron and Kerbosh, 1973] Bron, C. and Kerbosh, J. (1973). Algorithm 457 : Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9) :575–577.

[Busygin, 2002] Busygin, S. (2002). A new trust region technique for the maximum weight clique problem. *Submitted to Special Issue of Discrete Applied Mathematics : Combinatorial Optimization*.

[Carlier and Pinson, 1994] Carlier, J. and Pinson, E. (1994). Adjustments of heads and tails for the jobshop problem. *European Journal of Operational Research*, 78 :146–161.

[Caseau et al., 1993] Caseau, Y., Guillo, P.-Y., and Levenez, E. (1993). A deductive and object-oriented approach to a complex scheduling problem. In *Proceedings of DOOD'93*.

[Caseau and Laburthe, 1997] Caseau, Y. and Laburthe, F. (1997). Solving various weighted matching problems with constraints. In *Proceedings CP97*, pages 17–31, Austria.

[Chmeiss and Jégou, 1998] Chmeiss, A. and Jégou, P. (1998). Efficient path-consistency propagation. *International Journal on Artificial Intelligence Tools*, 7(2) :79–89.

[Codognet and Rossi, 2000] Codognet, P. and Rossi, F. (2000). Solving and programming with soft constraints : theory and practice. *Paper associated to the ECAI/AAAI00 tutorial on soft constraints*.

[Colmerauer, 1990] Colmerauer, A. (1990). An introduction to PROLOG III. *Communications of the ACM*, 33 :69–90.

[Dechter, 1992] Dechter, R. (1992). From local to global consistency. *Artificial Intelligence*, 55 :87–107.

[Dechter et al., 1991] Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint network. *Artificial Intelligence*, 49(1–3) :61–95.

[Dimacs, 1993] Dimacs (1993). Dimacs clique benchmark instances. *ftp ://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique*.

[Dubois et al., 1993] Dubois, D., Fargier, H., and Prade, H. (1993). The calculus of fuzzy restrictions as a basis for flexible constraint satisfaction. *FUZZ-IEEE'93*.

[Fahle, 2002] Fahle, T. (2002). Simple and fast : Improving a branch-and-bound algorithm for maximum clique. In Möring, R. and Raman, R., editors, *ESA 2002, 10th Annual European Symposium*, pages 485–498.

[Fargier et al., 1993] Fargier, H., Lang, J., and Schiex, T. (1993). Selecting preferred solutions in fuzzy constraint satisfaction problems. *First European Congress on Fuzzy and Intelligent Technologies*.

[Focacci et al., 1999a] Focacci, F., Lodi, A., and Milano, M. (1999a). Cost-based domain filtering. In *Proceedings CP'99*, pages 189–203, Alexandria, VA, USA.

[Focacci et al., 1999b] Focacci, F., Lodi, A., and Milano, M. (1999b). Integration of cp and or methods for matching problems. In *Proceedings CP-AI-OR 99*, Ferrara, Italy.

[Freuder, 1978] Freuder, E. (1978). Synthesizing constraint expressions. *CACM*, 21(11) :958–966.

[Freuder, 1989] Freuder, E. (1989). Partial constraint satisfaction. In *Proceedings IJCAI*, pages 278–283.

[Freuder and Wallace, 1992] Freuder, E. and Wallace, R. (1992). Partial constraint satisfaction. *Artificial Intelligence*, 58 :21–70.

[Gabrel et al., 1999] Gabrel, V., Knippel, A., and Minoux, M. (1999). Exact solution of multicommodity network optimization problems with general step cost functions. *Operations Research Letters*, 25 :15–23.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability : A guide to the theory of NP-completness*. W. H. Freeman, San Francisco.

[Gendron and Crainic, 1994] Gendron, B. and Crainic, T. (1994). Relaxations for multicommodity capacitated network design problems. Technical report, Centre de Recherche sur les Transports, Université de Montréal.

[Gervet, 1994] Gervet, C. (1994). Conjunto : constraint logic programming with finite set domains. In *Proceedings ILPS-94*.

[Golumbic, 1980] Golumbic, M. (1980). *Algorithmic Graph Theory and Perfect Graphs*. Academic Press.

[Gondran and Minoux, 1985] Gondran, M. and Minoux, M. (1985). *Graphes et Algorithmes*. Eyrolles, Paris.

[Henz et al., 2003] Henz, M., Müller, T., and Thiel, S. (2003). Global constraints for round robin tournament scheduling. *European Journal of Operational Research*, page To appear.

[Homer and Peinado, 1996] Homer, S. and Peinado, M. (1996). Experiements with polynomial-time clique approximation algorithms on very large graphs. In Johnson, D. and Trick, M., editors, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 26, pages 147–168. American Mathematical Society.

[Hopcroft and Karp, 1973] Hopcroft, J. and Karp, R. (1973). $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, 2 :225–231.

[ILOG, 1999] ILOG (1999). *ILOG Solver 4.4 User's manual*. ILOG S.A.

[Jan van Hoeve, 2004] Jan van Hoeve, W. (2004). A hyper-arc consistency algorithm for the soft alldifferent constraint. In *CP'04*, page to appear, Toronto, Canada.

[Kan Cheng et al., 2003] Kan Cheng, C., Ho Man Lee, J., and Stuckey, P. (2003). Box constraint collection for adhoc constraints. In *Proceedings CP'03*, pages 214–228, Kinsale, Ireland.

[Katriel and Thiel, 2003] Katriel, I. and Thiel, S. (2003). Fast bound consistency for the global cardinality constraint. In *Proceedings CP'03*, pages 437–451, Kinsale, Ireland.

[Larrosa et al., 1998] Larrosa, J., Meseguer, P., Schiex, T., and Verfaillie, G. (1998). Reversible DAC and other improvements for solving Max-CSP. *Proceedings AAAI*, pages 347–352.

[Larrosa and P.Meseguer, 1996] Larrosa, J. and P.Meseguer (1996). Exploiting the use of DAC in Max-CSP. *CP*.

[Laurière, 1976] Laurière, J.-L. (1976). *Un langage et un programme pour énoncer et résoudre des problèmes combinatoires*. PhD thesis, Université de Paris VI.

[Laurière, 1978] Laurière, J.-L. (1978). A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10 :29–127.

[Lawler, 1976] Lawler, E. (1976). *Combinatorial Optimization : Networks and Matroids*. Holt, Rinehart and Winston.

[Le Pape et al., 2002] Le Pape, C., Perron, L., Régin, J.-C., and Shaw, P. (2002). Robust and parallel solving of a network design problem. In *CP'02*, pages 633–648, Ithaca, NY, USA.

[Leconte, 1996] Leconte, M. (1996). A bounds-based reduction scheme for constraints of difference. In *Constraint-96, Second International Workshop on Constraint-based Reasoning*, Key West, FL, USA.

[Lecoutre et al., 2003] Lecoutre, C., Boussemart, F., and Hemery, F. (2003). Exploiting multidirectionnality in coarse-grained arc consistency algorithm. In *Proceedings CP'03*, pages 480–494, Cork, Ireland.

[Lemaitre et al., 2001] Lemaitre, M., Verfaillie, G., Bourreau, E., and Laburthe, F. (2001). Integrating algorithms for weigthed csp in a constraint programming framework. In *Soft'01, Modelling and Solving Problems with Soft Constraints*, Cyprus.

[Lopez-Ortiz et al., 2003] Lopez-Ortiz, A., Quimper, C.-G., Tromp, J., and van Beek, P. (2003). A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *IJCAI'03*, pages 245–250, Acapulco, Mexico.

[Mackworth, 1977] Mackworth, A. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8 :99–118.

[McAloon et al., 1997] McAloon, K., Tretkoff, C., and Wetzel, G. (1997). Sports league scheduling. In *Proceedings of ILOG user's conference*, Paris.

[Melhorn and Thiel, 2000] Melhorn, K. and Thiel, S. (2000). Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In *Proceedings of CP'00*, pages 306–319, Singapore.

[Micali and Vazirani, 1980] Micali, S. and Vazirani, V. (1980). An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings 21st FOCS*, pages 17–27.

[Mohr and Henderson, 1986] Mohr, R. and Henderson, T. (1986). Arc and path consistency revisited. *Artificial Intelligence*, 28 :225–233.

[Montanari, 1974] Montanari, U. (1974). Networks of constraints : Fundamental properties and applications to picture processing. *Information Science*, 7 :95–132.

[Östegard, 2003] Östegard, P. (2003). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, page to appear.

[Pesant, 2001] Pesant, G. (2001). A filtering algorithm for the stretch constraint. In *Proceedings CP'01*, pages 183–195, Pathos, Cyprus.

[Petit, 2002] Petit, T. (2002). *Modelization and Algorithms for solving over-constrained Problems*. PhD thesis, Université de Montpellier II.

[Petit et al., 2003a] Petit, T., Bessière, C., and Régin, J.-C. (2003a). Détection de conflits pour la résolution de problèmes sur-contraints. *Proceedings JNPC*, pages 293–307.

[Petit et al., 2003b] Petit, T., Bessière, C., and Régin, J.-C. (2003b). A general conflict-set based framework for over-constrained problems. *Proceedings of the 3th CP workshop on soft constraints*.

[Petit et al., 2000] Petit, T., Régin, J.-C., and Bessière, C. (2000). Meta constraints on violations for over-constrained problems. In *Proceedings ICTAI-2000*, pages 358–365.

[Petit et al., 2001] Petit, T., Régin, J.-C., and Bessière, C. (2001). Specific filtering algorithms for over-constrained problems. In *Proceedings CP'01*, pages 451–465, Pathos, Cyprus.

[Petit et al., 2002] Petit, T., Régin, J.-C., and Bessière, C. (2002). Range-based algorithms for max-csp. In *Proceedings CP'02*, pages 280–294, Ithaca, NY, USA.

[Puget, 1998] Puget, J.-F. (1998). A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of AAAI-98*, pages 359–366, Menlo Park, USA.

[Quimper et al., 2003] Quimper, C.-G., van Beek, P., López-Ortiz, A., Golynski, A., and Sadjad, S. (2003). An efficient bounds consistency algorithm for the global cardinality constraint. In *Proceedings CP'03*, pages 600–614, Kinsale, Ireland.

[Raymond, 2001] Raymond, E. (2001). *The Cathedral and the Bazaar*. O'Reilly.

[Régin, 1994] Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings AAAI-94*, pages 362–367, Seattle, Washington.

[Régin, 1995] Régin, J.-C. (1995). *Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la chimie organique*. PhD thesis, Université de Montpellier II.

[Régin, 1996] Régin, J.-C. (1996). Generalized arc consistency for global cardinality constraint. In *Proceedings AAAI-96*, pages 209–215, Portland, Oregon.

[Régin, 1997] Régin, J.-C. (1997). The global minimum distance constraint. Technical report, ILOG.

[Régin, 1998] Régin, J.-C. (1998). Modeling and solving sports league scheduling with constraint programming. In *INFORMS April 98*, Montreal, Canada.

[Régin, 1999a] Régin, J.-C. (1999a). Arc consistency for global cardinality with costs. In *Proceedings of CP'99*, pages 390–404, Alexandria, VA, USA.

[Régin, 1999b] Régin, J.-C. (1999b). The symmetric alldiff constraint. In *Proceedings of IJCAI'99*, pages 425–429, Stockholm, Sweden.

[Régin, 2002] Régin, J.-C. (2002). Cost based arc consistency for global cardinality constraints. *Constraints, an International Journal*, 7(3-4) :387–405.

[Régin, 2003] Régin, J.-C. (2003). *Constraints and Integer Programming combined*, chapter Global Constraints and Filtering Algorithms. M. Milano ed, Kluwer.

[Régin, 2003a] Régin, J.-C. (2003a). Solving the maximum clique problem with constraint programming. In *CP-AI-OR'03*, Montreal, Canada.

[Régin, 2003b] Régin, J.-C. (2003b). Using constraint programming to solve the maximum clique problem. In *CP'03*, pages 634–648, Kinsale, Ireland.

[Régin, 2004] Régin, J.-C. (2004). CAC : Un algorithme d'arc-consistance configurable, générique et adaptatif. In *JNPC'04*, Angers, France.

[Régin et al., 2000] Régin, J.-C., Petit, T., Bessière, C., and Puget, J.-F. (2000). An original constraint based approach for solving over constrained problems. In *Proceedings of CP'00*, pages 543–548, Singapore.

[Régin et al., 2001] Régin, J.-C., Petit, T., Bessière, C., and Puget, J.-F. (2001). New lower bounds of constraint violations for over-constrained problems. In *Proceedings CP'01*, pages 332–345, Pathos, Cyprus.

[Régin and Puget, 1997] Régin, J.-C. and Puget, J.-F. (1997). A filtering algorithm for global sequencing constraints. In *CP97, proceedings Third International Conference on Principles and Practice of Constraint Programming*, pages 32–46.

[Régin et al., 2002] Régin, J.-C., Puget, J.-F., and Petit, T. (2002). Representation of soft constraints by hard constraints. In *JFPLC*, pages 191–198, Nice, France.

[Régin and Rueher, 2000] Régin, J.-C. and Rueher, M. (2000). A global constraint combining a sum constraint and difference constraints. In *Proceedings of CP'00*, pages 384–395, Singapore.

[Rossi et al., 1990] Rossi, F., Petrie, C., and Dhar, V. (1990). On the equivalence of constraint satisfaction problems. In *Proceedings ECAI'90*, pages 550–556, Stockholm, Sweden.

[Rothlauf et al., 2002] Rothlauf, F., Goldberg, D., and Heinzl, A. (2002). Network random keys : A tree representation scheme for genetic and evolutionary algorithms. *Evolutionary Computation*, 1(10) :75–97.

[Schiex, 1992] Schiex, T. (1992). Possibilistic constraint satisfaction problems, or "how to handle soft constraints ?". In *Proceedings of 8th Conf. of Uncertainty in AI.*

[Schiex et al., 1995] Schiex, T., Fargier, H., and Verfaillie, G. (1995). Valued constraint satisfaction problems : hard and easy problems. *Proceedings IJCAI.*

[Schiex et al., 1997] Schiex, T., Fargier, H., and Verfaillie, G. (1997). Problèmes de satisfaction de contraintes valués. *Revue d'Intelligence Artificielle*, 11.

[Schiex et al., 1996] Schiex, T., Régin, J.-C., Gaspin, C., and Verfaillie, G. (1996). Lazy arc consistency. In *AAAI-96, proceedings Thirteenth National Conference on Artificial Intelligence*, pages 216–221, Portland, Oregon.

[Sellmann, 2003] Sellmann, M. (2003). Approximated consistency for knapsack constraints. In *Proceedings CP'03*, pages 679–693, Kinsale, Ireland.

[Simonis, 1996] Simonis, H. (1996). Problem classification scheme for finite domain constraint solving. In *CP96, Workshop on Constraint Programming Applications : An Inventory and Taxonomy*, pages 1–26, Cambridge, MA, USA.

[St-Louis et al., 2003] St-Louis, P., Gendron, B., and Ferland, J. (2003). A penalty-evaporation heuristic in a decomposition method for the maximum clique problem. In *Optimization Days*, Montreal, Canada.

[Stergiou and Walsh, 1999] Stergiou, K. and Walsh, T. (1999). The difference all-difference makes. In *Proceedings IJCAI'99*, pages 414–419, Stockholm, Sweden.

[Tarjan, 1983] Tarjan, R. (1983). *Data Structures and Network Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics.

[Truchet et al., 2001] Truchet, C., Agon, C., and Assayag, G. (2001). Recherche adaptative et contraintes musicales. *Neuvièmes Journées Francophones de Programmation Logique et de Programmation par Contraintes (JFPLC)*.

[van Dongen, 1997] van Dongen, M. (1997). Ac-3b, an efficient arc-consistency algorithm with low space-complexity. Technical Report TR-97-01, Department of Computer Science, National University of Ireland, Cork, College Road, Cork, Ireland.

[van Dongen, 2002] van Dongen, M. (2002). Ac-3d an efficient arc-consistency algorithm with a low space-complexity. In *Proceedings CP'02*, pages 755–760, Ithaca, NY, USA.

[van Dongen, 2002] van Dongen, M. (2002). *Constraints, Varieties and Algorithms*. PhD thesis, Department of Computer Science, University College Cork.

[van Dongen and Bowen, 2000] van Dongen, M. and Bowen, J. (2000). Improving arc-consistency algorithms with double-support checks. In *Proceedings of the Eleventh Irish Conference on Artificial Intelligence and Cognitive Science (AICS'2000)*, pages 140–149.

[Van Hentenryck, 1989] Van Hentenryck, P. (1989). *Constraint Satisfaction in Logic Programming*. M.I.T. Press.

[Van Hentenryck and Deville, 1991] Van Hentenryck, P. and Deville, Y. (1991). The cardinality operator : A new logical connective for constraint logic programming. In *Proceedings of ICLP-91*, pages 745–759, Paris, France.

[Van Hentenryck et al., 1992] Van Hentenryck, P., Deville, Y., and Teng, C. (1992). A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57 :291–321.

[Van Hentenryck et al., 1999] Van Hentenryck, P., Michel, L., L.Perron, and Régin, J.-C. (1999). Constraint programming in opl. In *PPDP 99, International Conference on the Principles and Practice of Declarative Programming*, pages 98–116, Paris, France.

[van Hentenryck et al., 1998] van Hentenryck, P., Saraswat, V., and Deville, Y. (1998). Design, implementation, and evaluation of the constraint language cc(fd). *Journal of Logic Programming*, 37(1–3) :139–164.

[Verfaillie, 1997] Verfaillie, G. (1997). *Méthodes génériques pour l'optimisation sous contraintes*. Habilitation à diriger les recherches, synthèse des travaux. Onera-Cert.

[Wallace, 1994] Wallace, R. (1994). Directed arc consistency preprocessing as a strategy for maximal constraint satisfaction. *Proceedings ECAI*, pages 69–77.

[Waltz, 1975] Waltz, D. L. (1975). Understanding line drawings of scenes with shadows. In *The Psychology of Computer Vision*, pages 19–91. McGraw Hill. d'abord paru dans Tech. Rep AI271, MIT MA, 1972.

[Wood, 1997] Wood, D. (1997). An algorithm for finding maximum clique in a graph. *Operations Research Letters*, 21 :211–217.

[Zhang and Yap, 2001] Zhang, Y. and Yap, R. (2001). Making ac-3 an optimal algorithm. In *Proceedings of IJCAI'01*, pages 316–321, Seattle, WA, USA.

[Zhou, 1996] Zhou, J. (1996). A constraint program for solving the job-shop problem. In *Proceedings of CP'96*, pages 510–524, Cambridge.

[Zhou, 1997] Zhou, J. (1997). *Computing Smallest Cartesian Products of Intervals : Application to the Jobshop Scheduling Problem*. PhD thesis, Université de la Méditerranée, Marseille.