Dossier de Qualification aux fonctions de Professeur des Universités

27ème section du CNU

Jean-Charles Régin

Table des matières

Synthèse du dossier

Formation

• **Habilitation à Diriger des Recherches** : *"Modélisation et Contraintes Globales en Programmation par Contraintes"*, Univ. Nice-Sophia Antipolis, Nov. 2004. **Qualification aux fonctions de professeur des universités section 27** obtenue en 2005.

• **Doctorat Informatique** : "*Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la chimie organique*", Univ. Montpellier II, Déc. 1995. Mention très honorable avec félicitations du jury.

• **D.E.A. Informatique**, Univ. Montpellier II, 1990.

Activités professionnelles

• Depuis Oct. 08 : **Professeur Associé, Université de Nice-Sophia Antipolis, Ecole Polytechnique Universitaire**

• Mars 01 - Déc 05 et Août 06 - Sept. 08 : **Directeur de la Programmation par Contraintes (PPC) à ILOG**. Gestion d'une équipe de 7 personnes dont 4 docteurs ; gestion de 2 projets nationaux et de 2 projets internationaux pour un montant total de subvention égal à €750,000 ; gestion des produits ILOG Solver et JSolver (moteurs de résolution de problèmes d'optimisation combinatoire).

• Janv. 05 - Juil. 05 : **Chercheur à Cornell University, USA**

• Juil. 98 - Fév. 01 : **Chef de Projet ILOG Solver**

• Janv. 96 - Juin 98 : **Ingénieur de développement ILOG Solver**

Enseignement

Enseignement à l'Univ. Nice-Sophia Antipolis en 08/09 :

• Programmation Concurrente (en Java) : 40h

• Logique : 42h

• Programmation par Contraintes : 24h

• Gestion de Projets en 4ème année : Etude et Analyse du système actuel et proposition de nouvelles modalités qui répondent mieux aux demandes méthodologiques industrielles.

Formation continue :

• J'ai assuré la formation continue des consultants ou des ingénieurs d'avant-vente d'ILOG en ce qui concerne les nouveautés et l'utilisation des produits ILOG dont j'ai eu la charge (20h/an pendant 12 ans).

Interventions sur la Programmation par Contraintes, cours ou séminaires de 3 ou 6h depuis 97 :

• Ecole Polytechnique de Paris.

- Ecole Centrale de Paris.
- DEA, Univ. Nice-Sophia Antipolis et Univ. Montpellier II.
- Greco Informatique.

Activités Scientifiques

Activité de recherche actuelle :

- Ecriture d'un livre de synthèse sur la programmation par contraintes et sur la méthodologie de résolution d'applications.
- Développement d'algorithmes permettant de faciliter et d'accélérer la résolution de problèmes en PPC.

Publications

- J'ai publié : 2 chapitres de livres, 5 articles dans des journaux internationaux, 27 articles dans des conférences internationales avec comité de lecture et hautement sélectives (30 % d'acceptation).
- J'ai été invité à donner 9 présentations dans des conférences et workshops internationaux et 8 séminaires dans des Universités mondialement reconnues (Brown, Cornell, Montreal, Georgia Tech, ...)
- Hirsch-index égal à 21.
- Je suis l'auteur d'un des articles les plus cités de mon domaine (plus de 450 références sur Google Scholar).

Activités éditoriales :

- Editeur en chef (avec P. Van Hentenryck) du journal international Constraint Programming Letters.
- Membre du comité d'organisation de la conférence internationale CPAIOR.
- Membre du comité de lecture du journal international Constraints.
- Chairman et organisateur (avec M. Rueher) de CPAIOR'04.
- Organisateur de 2 workshops internationaux "Non Binary Constraints".
- Membre des comités de programme des conférences internationales : IJCAI-09, AAAI-08 (senior member), AAAI-05, CPAIOR (depuis 04), CP (06, 05, 03, 01)...

Encadrement de thèses de Doctorat et participation à des jurys :

- T. Petit, Univ. Montpellier II, soutenue le 29 Nov 02, Co-Directeur.
- D-O. Fernandez-Pons, Univ. Paris VI, depuis 04, Co-Directeur.
- P. Schaus, Univ. Louvain, Belg., depuis 06, Membre du comité d'encadrement.
- Rapporteur d'une HDR, de 2 thèses de Doctorat et membre du jury d'une HDR et de 5 thèses de Doctorat

Projets :

Participation :

- Depuis Oct. 08 : MANCOOSI, Communauté Européenne (€430,000)

Participation et Gestion :

- Fév. 03 - Déc. 04 : FADO, Ministère de l'Industrie (€240,000)
- Juin 03 - Juin 04 : ORINCP, US Air Force ($25,000).
- Juin 99 - Sept. 02 : ECSPLAIN, Communauté Européenne (€320,000)
- Mars 99 - Mai 01 : ROCOCO, Ministère des Télécom. (€150,000)

Page web

www.constraint-programming.com/people/regin
Copie du dossier : www.constraint-programming.com/people/regin/papers/qualif08.pdf

Chapitre 1

Curriculum Vitae

Jean-Charles Régin

Hameau de la Palmeraie, Bâtiment D2,
934, chemin des Ames du Purgatoire,
06600 Antibes – FRANCE
Tel : +33 4 93 74 49 85

Université de Nice - Sophia Antipolis,
Ecole Polytechnique Universitaire,
930 Route des Colles - BP 145 06903 Sophia Antipolis Cedex France
Tél. +33 4 92 94 27 59 - Fax : +33 4 92 94 28 98
courriel : regin@polytech.unice.fr

né à Mulhouse (France), le 11 Janvier 1966
Nationalité française
Marié, deux enfants

1.1 Formation

- **Habilitation à diriger des recherches** :
  "Modélisation et Contraintes Globales en Programmation par Contraintes",
  Université de Nice-Sophia Antipolis, 16 Novembre 04,
  Directeur : Michel Rueher, Professeur, Univ. Nice-Sophia Antipolis ;
  Rapporteurs :
  • Jacques Carlier, Professeur, Univ. de Technologie de Compiègne,
  • Eugene Freuder, Professeur, Univ. College Cork, Ireland,
  • Pascal Van Hentenryck, Professeur, Brown Univ., USA ;
  Jury :
  • Yves Caseau, Directeur des Sytèmes d'Information, Bouygues Telecom,
  • Jacques Carlier, Professeur, Univ. de Technologie de Compiègne,
  • Alain Colmerauer, Professeur Univ. Méditerranée,
  • Michel Cosnard, Directeur de l'Unité de Recherche INRIA Sophia Antipolis, Président,
  • François Fages, Directeur de Recherche, INRIA Rocquencourt,
  • Michel Rueher, Professeur, Univ. Nice-Sophia Antipolis
  Qualification aux fonctions de professeur des universités section 27 obtenue en 2005.

– **Doctorat Informatique** :
"*Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la chimie organique*",
Université de Montpellier II, Décembre 95.
Mention très honorable avec félicitations du jury.
Directeur de thèse : Olivier Gascuel, Chercheur CNRS, LIRMM Montpellier
Rapporteurs :
  • Yves Deville, Professeur, Univ. Catholique de Louvain, Belgique,
  • Amedeo Napoli, Chercheur CNRS, CRIN Nancy,
  • Michel Rueher, Professeur, Univ. Nice-Sophia Antipolis,

Jury :
  • Christian Bessière, Chercheur CNRS, LIRMM Montpellier,
  • Bertrand Castro, Directeur des activités chimiques, Sanofi,
  • Olivier Gascuel, Chercheur CNRS, LIRMM Montpellier,
  • Michel Habib, Professeur, Univ. Montpellier II, Président,
  • Claude Laurenço, Directeur de Recherche CNRS, LIRMM Montpellier,
  • Amedeo Napoli, Chercheur CNRS, CRIN Nancy,
  • Jean-François Puget, Chef de Projet, ILOG,
  • Joël Quinqueton, Directeur de Recherche INRIA, LIRMM Montpellier,
  • Michel Rueher, Professeur, Univ. Nice-Sophia Antipolis

– **D.E.A. Informatique** :
"*Apprentissage des liaisons stratégiques en chimie organique*",
Université de Montpellier II, 90.
Directeur : Olivier Gascuel, Chercheur CNRS, LIRMM Montpellier.

1.2 Activités professionnelles

– *Depuis Oct. 08* : **Professeur Associé, Université de Nice-Sophia Antipolis, Ecole Polytechnique Universitaire, Département Sciences Informatiques (Ex ESSI)**.
  • En Enseignement : j'assure des cours de Logique, Programmation par Contraintes (PPC) et Programmation Concurrente. Je profite de mon expérience passée et de mon regard extérieur pour modifier les projets des étudiants de l'Ecole Polytechnique afin de mieux répondre aux demandes méthodologiques industrielles.
  • En Recherche : je participe à un projet européen, je travaille à l'amélioration d'algorithmes de filtrage en PPC. Enfin, j'écris un livre sur la modélisation et la résolution de problèmes complexes en PPC.

– *Mars 01 à Déc. 05 et d'Août 06 à Sept.08* : **Directeur de la Programmation par Contraintes à ILOG**, avec quatre activités principales :
  • Recherche en programmation par contraintes. Développement d'algorithmes originaux afin d'améliorer la résolution de problèmes réels complexes par la PPC. Intégration d'algorithmes venant de domaines divers (recherche local, recherche opérationnelle). Travail sur la modélisation de problèmes : construction de modèles efficaces en pratique, réflexion sur la modélisation pour produire des règles pouvant être utilisées par des personnes non expertes du domaine. Encadrement de thésards.
  • Gestion des produits ILOG Solver et ILOG JSolver, qui sont des moteurs de résolution de problèmes d'optimisation basé sur la programmation par contrainte, et des librairies ILOG Concert et ILOG JConcert ;
  • Gestion d'une équipe de sept personnes dont deux chefs de projet ;
  • Gestion de projets nationaux ou européens.

- *Janv. 05 - Juil. 05* : **Chercheur à Cornell University, USA**, Membre du Département "Computing and Information Science".
- *Juil. 98 - Fév. 01* : **Chef de Projet ILOG Solver à ILOG**.
- *Janv. 96 - Juin 98* : **Ingénieur de Développement ILOG Solver à ILOG**.
- *Juin 91 - Dec. 95* : **Doctorant au LIRMM** (Laboratoire d'Informatique de Robotique et de Micro-électronique de Montpellier).
  Ce travail a été financé par SANOFI-CHIMIE et le CNRS.
- *Dec. 90 - Juin 91* : **Vacataire au CNRS**.
  J'ai travaillé sur la conception et l'implémentation du système RESYN : un système de planification de rétrosynthèse en chimie organique.

1.3 Récompense

Le système RESYN, dont je suis un coauteur, a reçu en 1993 le premier prix (50 000 FF) de l'inovation et de la recherche délivré par l'ADER Languedoc Roussillon (Association pour le Développement de l'Enseignement et de la Recherche).

Chapitre 2

Enseignement

Etant depuis peu en poste (depuis Octobre) à l'Université de Nice-Sophia Antipolis, je commence à donner de façon classique des cours. Lorsque je faisais partie du monde industriel, j'ai toujours tenu à faire de l'enseignement. C'était important pour moi, car cela me permettait d'être au contact de futurs collaborateurs et de transmettre les connaissances du domaines sous une forme que l'on peut améliorer au fur et à mesure des cours. Je reste particulièrement intéressé par l'enseignement de la programmation par contraintes (PPC), car c'est un domaine jeune qui ne demande qu'à être simplifié.

2.1 Enseignement à l'Université de Nice-Sophia Antipolis en 08/09

- Programmation Concurrente (en Java) : 40h
- Logique : 42h
- Programmation par Contraintes : 24h

Au titre de mon complément de service, j'ai été chargé par M. Riveill, Directeur du Département Sciences Informatique, d'étudier comment certaines remarques des futurs employeurs à propos de la formation des étudiants pourraient être mieux prises en compte. Notamment, il apparait que les étudiants ont très fréquemment du mal à respecter les règles en vigueur dans les entreprises, par exemple les règles d'écriture de code. Je suis donc chargés de modifier les projets que les étudiants doivent réalisés pendant leur cursus afin de mieux prendre en compte ce type de problème.

2.2 Formation continue

J'ai assuré la formation continue des membres du département Recherche et Développement à ILOG depuis 1996, afin de faire connaître les différents travaux de recherche actuel en programmation par contraintes. J'ai également effectué la formation continue des consultants ou des ingénieurs d'avant-vente d'ILOG en ce qui concerne les nouveautés et l'utilisation des produits ILOG dont j'ai eu la charge. Cela représentant un volume de cours d'environ 240 heures (20h/an pendant 12 ans).

2.3 Interventions sur la PPC, cours ou séminaires de 3 ou 6h depuis 97

- Ecole Polytechnique de Paris.
- Ecole Centrale de Paris.
- DEA de l'Université de Nice-Sophia Antipolis.
- DEA de l'Université de Montpellier II.
- Greco Informatique.

2.4 Tutoriaux et Ecoles d'été

J'ai donné trois tutoriaux pendant les conférences majeures de PPC :

- J-C. Régin, "Modeling Problems in Constraint Programming", **CP'04**, Toronto, Canada, Sept. 04.
- N. Beldiceanu and J-C. Régin, "Global Constraints", **CP'02**, Ithaca, USA, Sept. 02.
- J-C. Régin, "Global Constraints", Tutorial invité, **CP-AI-OR'02**, Le Croisic, France, Mai 02.

Je suis intervenu pour donner des cours dans trois écoles d'été :

- J-C. Régin, "Alldifferent and Cardinality Constraints", **Second International Summer School on Constraint Programming**, Samos, Greece, July 2006.
- J-C. Régin, "Global Constraints", **First International Summer School on Constraint Programming**, Aquafredda di Maratea, Italy, Sept 2005.
- J-C. Régin, "Graph Theory and Constraint Programming", Master Class, **CP-AI-OR'04**, Nice, France, Avril 2004.

Certains transparents de ces tutoriaux sont disponibles sur le web (http ://ai.uwaterloo.ca/ cp2004/tutorials.html, www-sop.inria.fr/coprin/cpaior04/, http ://www.math.unipd.it/ frossi/cp-school/reginslides-summerSchoolTalk.pdf).

2.5 Documents Pédagogiques

La documentation d'un produit ILOG est composée de deux parties : un manuel de référence et un manuel utilisateur. Dans le cas d'ILOG Solver, le manuel utilisateur s'apparente fortement à un cours de PPC. Jusqu'en 1999 la documentation était réalisé par les développeurs. J'ai écrit trois chapitres généraux et trois chapitres décrivant des exemples d'utilisation du produit. A partir de 1999, des documentalistes ont été recrutées. La documentation est depuis écrite par ces personnes sous le contrôle des développeurs.

J'ai également été fortement impliqué dans la conception et la rédaction des cours associés à ILOG Solver.

2.6 Projet d'Enseignement

J'ai le désir et la volonté de faire bénéficier les étudiants de l'expérience que j'ai acquise en programmation et en génie logiciel. Je voudrais notamment montrer les différences importantes qui existe entre une interface utilisateur, la structure interne du code (par exemple la hiérarchie de classes dans un langage à objet) et les problèmes d'efficacité de l'implémentation. Il me semble important de bien différencier la théorie de l'application. Par exemple, la théorie est principalement basée sur la complexité dans le pire des cas. Or, il arrive fréquemment que l'évaluation du pire des cas soit sur-estimé faute de mieux. On peut donc avoir un algorithme

qui théoriquement est moins bon, mais qui s'avère réellement plus performants en pratique.

J'aimerais aussi simplifier la présentation de certains domaines comme les flots dans la théorie des graphes. Tout d'abord, de nombreux algorithmes sont beaucoup plus simples si l'on se contente d'étudier les flots ne pouvant prendre que des valeurs entières. Vu qu'il est difficile de représenter d'autres objets discrets que des rationnels dans un ordinateur actuel, cette approche me semble pertinente. Enfin, il me semble important de commencer par expliquer conceptuellement les algorithmes avant de manipuler essentiellement des équations algébriques. On peut tout à fait comprendre presque tous les algorithmes de recherche de flots à coût minimum en introduisant uniquement à la fin la notion de coûts réduits. Cette notion permet d'éviter d'avoir des coûts négatifs sur les arcs et donc d'améliorer la complexité d'algorithmes de plus courts chemins, mais elle complexifie la compréhension des algorithmes. Mon projet est donc de présenter de façon plus pédagogique de nombreux domaines de l'informatique et plus particulièrement l'algorithmique.

Je souhaiterais également pouvoir enseigner la PPC à des étudiants en gestion. La programmation linéaire est bien connue dans ce milieu alors qu'elle est, à mon avis, moins intuitive et plus complexe à mettre en oeuvre que la PPC. J'aimerais donc montrer aux futurs managers qu'une technique de résolution de problèmes complexes comme la PPC mérite d'être connue.

Enfin, de part mon expérience professionnelle, j'ai des compétences en langage de programmation et en environnement de programmation, et je pourrais effectuer ces enseignements de base.

Chapitre 3

Recherche

3.1 Activité de Recherche actuelle et Projet

La société ILOG, dans laquelle j'ai travaillé pendant 12 ans, a toujours privilégié les communications lors de conférence plutôt que les articles dans les revues. En particulier, la Direction Recherche et Développement d'ILOG ne souhaitait pas que le personnel consacre du temps à la rédaction d'une "version journal" d'un article déjà publié dans les actes d"une conférence internationale reconnue. Ayant quitté la société, j'écris actuellement des versions révisées et étendues de certains de mes papiers. Je travaille également sur un chapitre décrivant l'ensemble des travaux de recherches faits depuis 10 ans sur les contraintes globales, pour lequel M. Milano et P. Van Hentenryck, éditeurs, m'ont sollicité.

Cependant, je consacre principalement mon activité de Recherche à l'écriture d'un livre de synthèse sur la programmation par contraintes et sur la méthodologie de résolution d'applications réelles. Ce livre s'adresse à tout type de personne qui veut résoudre un problème réel avec la PPC. Il s'agit à la fois d'un guide pour la résolution d'application réelle s'adressant aux gens ayant peu d'expérience dans ce domaine, et d'une sorte de livre de recettes pour utilisateurs plus confirmés (ce que les anglosaxons appellent un cookbook) afin de les aider à améliorer la représentation et la résolution de leurs problèmes. Ce type de livre n'existe pas en PPC. J'essaie d'avoir un point de vue original et assez exhaustif de toutes les difficultés et solutions pour résoudre des problèmes complexes que j'ai rencontrées ou dont on m'a fait part lorsque je travaillais dans l'industrie.

Le plan actuel de ce livre est fourni en tant que publication jointe.

Enfin, je suis intégré à l'équipe Contraintes et Preuves (CeP) du laboratoire I3S je participe actuellement au projet de recherche européen MANCOOSI dont cette équipe est un des partenaires. Ce projet a pour but de simplifier les mises à jour des logiciels open-sources afin de les rendre plus transparentes.

Projet de Recherche Mon objectif est de rendre la programmation par contraintes (PPC) plus efficace et plus simple à utiliser pour résoudre ces nombreux problèmes difficiles en optimisation combinatoire qui constituent encore de véritables challenges pour notre communauté. Je m'investirai ainsi tout particulièrement dans le développement de contraintes globales et des algorithmes de filtrage associés. Ces algorithmes, basés sur la théorie des graphes ou la recherche opérationnelle, identifient très tôt des valeurs qui n'appartiennent à aucune solution du problème considéré. Cette approche qui a été peu utilisée en dehors de la PPC pourrait donner naissance à un nouveau thème de recherche transversal regroupant des chercheurs issus de diverses communautés (PPC, RO, algorithmique ...). 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 est une approche récente et prometteuse qui mérite d'être approfondie.

Pour rendre la PPC plus simple à utiliser et pour faciliter sa mise en oeuvre sur les problèmes d'optimisation, je travaillerai sur l'intégration de coûts à l'intérieur même des contraintes ainsi que sur la définition de nouvelles contraintes globales molles.

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. C'est pourquoi, j'aimerais continuer l'étude de la résolution de problèmes difficiles purs, autrement dit continuer le travail que j'ai commencé avec le problème de la recherche de la taille de la plus grande clique dans un graphe.

3.2 Description détaillée des Thèmes de Recherche

Mes thèmes de recherche concernent essentiellement la programmation par contraintes (PPC) :

- **Modélisation**. Ce thème consiste à identifier les avantages et les inconvénients d'un modèle, afin d'exhiber des règles générales sur la qualité et l'efficacité supposée d'un modèle. L'un des buts est aussi la mise à disposition de l'utilisateur d'outils lui permettant de définir plus facilement un bon modèle.
- **Contraintes**. Il s'agit de développer :
  - Des algorithmes génériques permettant de développer facilement, voir automatiquement, des algorithmes de filtrage [1] associés aux contraintes.
  - Des contraintes globales spécifiques, c'est-à-dire utilisant la structure des contraintes afin d'être plus efficace en terme de réduction de domaine, mais aussi en temps. C'est notamment dans ce cadre que de nombreux algorithmes de Recherche Opérationnelle sont intégrés à la PPC.
  - Des contraintes flexibles, c'est-à-dire étant capable d'intégrer d'autres contraintes dont la structure n'est pas connu à-priori.
- **Problèmes sur-contraints**. Il s'agit de proposer des méthodes génériques, réalistes et couvrant les besoins des utilisateurs. Le développement de contraintes molles associées à des algorithmes de filtrage dédiés fait partie de ce thème.
- **Résolution de problèmes réels**. Plusieurs sous-thèmes se distinguent, comme l'identification de la difficulté c'est-à-dire la recherche de sous-ensembles de contraintes et de variables qui forment un sous-problème difficile à résoudre ; ou encore l'étude de méthodes pouvant traiter des problèmes de très grandes tailles (plusieurs milliers de variables et/ou de contraintes). Parmi les types de problèmes considérés on peut citer : la recherche de l'existence d'un isomorphisme de sous-graphe entre deux graphes, la recherche de la taille de la plus grande clique dans un graphe, l'ordonnancement des voitures sur une chaîne de montage, le dimensionnement de réseau de télécommunication, la détermination du calendrier de compétitions sportives ou le remplissage de carrés latins sous contraintes.

1. Un algorithme de filtrage associé à une contrainte C supprime des valeurs des domaines des variables qui ne satisfont pas C.

3.3 Résultats

Ma contribution personnelle au domaine de la PPC peut se résumer aux trois points suivants :

- de nombreuses contraintes globales fondamentales associées à des algorithmes de filtrage originaux ou intégrant des algorithmes de Recherche Opérationnelle. J'ai écrit le premier algorithme de filtrage établissant la consistance d'arc pour une contrainte globale. L'article associé est un des articles les plus cités en programmation par contraintes (voir Regin, AAAI-94). Tous les résolveurs basés sur la PPC intègrent la plupart de mes travaux sur les contraintes globales.
- divers principes généraux de modélisation et plus particulièrement un nouveau modèle pour la résolution des problèmes sur-contraints, accompagnés de plusieurs algorithmes de filtrage améliorant l'existant et 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, comme la recherche de la taille de la plus grande clique dans un graphe ou le dimensionnement de réseau de télécommunication.

La diffusion de mes travaux dépasse largement mon domaine. Mes travaux sont, par exemple, connus en Recherche Opérationnelle. J'ai notamment été personnellement invité par Georges Nemhauser à Atlanta (Georgia Tech University), Michel Gendreau à Montréal (Université de Montréal) et Pascal Van Hentenryck à Providence (Brown University). J'ai également été invité à présenter mes travaux dans des centres de recherche industriel comme IBM Watson Research Center.

3.4 Gestion de projets

La participation à des projets nationaux ou internationaux est un moyen efficace pour collaborer avec d'autres industriels et pour maintenir des contacts universitaires. C'est pourquoi, je me suis toujours efforcé d'être fortement impliqués dans divers projets. En voici la liste :

1. Depuis Oct. 08 : participant pour l'Université de Nice-Sophia Antipolis au projet **MANCOOSI** : "MAnaging the COmplexity of Open Source Infrastructure". Ce projet est un projet de recherche financé par la Communauté Européenne (FP7-ICT-Challenge 1-Objective 2007.1.2). La subvention de l'Université est de 430,000 euros.

   Le but est de résoudre le problème de la complexité des mises à jour des logiciels. Pour cela, MANCOOSI vise à développer des algorithmes spécifiques afin d 'améliorer la bonne marche des mises à jour et de permettre, en cas d'échec, un retour en arrière. Le programme espère ainsi que les mises à jour logicielles poseront moins de problèmes et seront davantage transparentes, tant pour l'utilisateur débutant que pour le professionnel.

2. Fév. 03 - Déc 04 : responsable ILOG du projet **FADO** : "Faciliter l'hybridation des algorithmes d'optimisation combinatoire par des contraintes hétérogènes spécifiques à un métier ou une application sans sacrifier la performance". Ce projet est financé par le ministère de l'industrie (subvention ILOG 240,000 euros).

   L'objectif de ce projet est d'étudier les différents moyens de "faire descendre" les contraintes additionnelles dans les algorithmes de bases, lorsque cela est possible. Autrement dit, il s'agit de déterminer s'il est possible d'intégrer une base d'algorithmes "ouverts" au sein de la suite d'outils d'optimisation et de

développer un outil logiciel générique pour exploiter ces algorithmes. Ceci afin de permettre à de nombreux clients d'aborder plus simplement des problèmes d'optimisation qui restent aujourd'hui difficiles à résoudre.

3. Juin 2003 à Juin 2004 : "Principal investigator" du projet "Integrating OR algorithms and randomization with constraint programming", EOARD-AFOSR FA8655-03-1-3022, financé par European Office of Aerospace Research and Developpement : a detachment of the Air Force of Scientific Research (subvention ILOG $25,000). Ce financement est le premier que l'EOARD ait attribué à une entreprise française d'informatique.

   L'objectif de ce projet est le développement d'algorithmes nouveaux en PPC basés sur des algorithmes de Recherche Opérationnelle.

4. Mars 1999-Mai 2001 : responsable ILOG du projet **ROCOCO** de Mars 1999 Mai 2000 : "Recherche Opérationnelle et Contraintes pour la Conception de Réseau". Ce projet a été financé par le ministère des télécommunication (subvention ILOG 150 000 euros). J'ai réalisé la partie PPC de ce projet, puis C. Le Pape m'a remplacé comme responsable ILOG du projet.

   L'objectif du projet ROCOCO est de concevoir des algorithmes fondés sur l'utilisation conjointe de techniques de Recherche Opérationnelle (RO) et de Programmation Par Contraintes (PPC) pour résoudre des problèmes de conception et d'optimisation de réseaux de télécommunication, réseaux d'entreprises en particulier.

5. Juin 1999-Sept. 2002 : responsable ILOG du projet européen **ECSPLAIN** : "Exploiting non standard CSP for Leveraging Application Intelligence". Ce projet a été financé par la communauté européenne (subvention ILOG 320 000 euros).

   L'objectif de ce projet est de développer des méthodes, techniques et logiciels génériques pour résoudre les problèmes industriels sur-contraints impliquant des critères d'optimisation multiple et/ou une grande variété de contraintes de préférence. Ce projet permet de représenter explicitement des problèmes industriels complexes, autorisant des requêtes de haut niveau, des préférences et des critères d'optimisation multiples. Les problèmes considérés sont la gestion des forêts, et l'ordonnancement d'activités notamment lors de la construction du métro de Casablanca.

3.5 Encadrements et Jurys

Doctorat

- J'ai été codirecteur (avec Christian Bessière) de la thèse de Thierry Petit : "Modélisation et Algorithmes de Résolution de Problèmes Sur-Contraints", soutenue le 29 Novembre 2002 à l'Université de Montpellier II. Thierry Petit est actuellement Maître Assistant à l'école des Mines de Nantes.
- Je suis actuellement (depuis 04) codirecteur (avec Michel Minoux) de la thèse de Diego Olivier Fernandez Pons, Université de Paris VI Pierre et Marie Curie.
- Je suis membre (depuis 06) du comité d'encadrement de la thèse de Pierre Schaus, Université de Louvains-la-Neuve, Belgique.

DEA

J'ai encadré le stage de DEA de Christophe Fagot en 95, ainsi que celui de Thierry Petit en 98. DEA soutenus à l'Université de Montpellier II.

Rapport de Thèses et Participation à des Jurys

J'ai été rapporteur d'une HDR et de 2 thèses de Doctorat. J'ai été membre du jury d'une HDR et de 5 thèses de Doctorat.

3.6 Activités d'Animation Scientifiques

Responsabilité Editoriales

- Co-editeur avec Pascal Van Hentenryck du journal électronique Constraint Programming Letters.
- Membre du comité d'édition du journal international Constraints Journal.

Animation de la Communauté

- Membre du comité d'organisation de la conférence internationale CP-AI-OR.
- Membre du comité de pilotage de l'option GIPAD, Ecole des Mines de Nantes, 07.
- Membre élu du comité exécutif de l'"Association for Constraint Programming" en 06.
- Membre élu du conseil d'administration de l'Association Française pour la Programmation par Contraintes de 04 à 08.

Organisation de Conférences et Workshops Internationaux

- Chairman et Organisateur (avec Michel Rueher) de CP-AI-OR'04, First International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems.
- Chairman du workshop "Non Binary Constraints" qui s'est tenu pendant la conférence IJCAI-99.
- Chairman du workshop "Non Binary Constraints" qui s'est tenu pendant la conférence ECAI-98.

Participations à des comités de programmes de conférences

Conférences Internationales :
IJCAI : 09
AAAI : 08 (senior member), 05
CP-AI-OR : 09, 08, 07, 06, 05, 04
CP : 06, 05, 03, 01.

Conférences Nationales :
JFPC'07, JFPC'06, JFPC'05, JNPC'04, JFPLC'02, RNJCIA'98, JNPC'97.

Relectures

Depuis 1995 je suis relecteur régulier pour les revues : Artificial Intelligence, Constraint Journal, JAIR. Je participe également à l'évaluation d'articles pour les conférences : CP, CP-AI-OR, AAAI, IJCAI, ECAI, JNPC, JFPLC.

3.7 Collaborations externes et Séjours scientifiques

J'ai collaboré sous diverses formes (relations contractuelles, visites, séminaires...) avec de nombreux chercheurs : Christian Bessière (LIRMM, France), Yves Deville (Université de Louvain-la-Neuve, Belgique), Carla Gomes (Cornell University, USA), Eugene Freuder (Cork University, Ireland), Christine Gaspin (INRA Toulouse), Gilles Pesant (Université Montréal, Canada), Pierre Schaus (Université de Louvain-la-Neuve, Belgique), Thomas Schiex (INRA Toulouse), Michel Rueher

(Université de Nice Sophia-Antipolis), Pascal Van Hentenryck (Brown University, USA).
Ces collaborations ont toutes débouché sur des publications.

Séjours scientifiques invités

- Université de Nantes, Fév 09, 1 semaine.
- Brown University, Déc. 08, 1 semaine.
- Cornell University, Janv. 05 - Juil. 05, 7 mois.
- Cornell University, Avr. 04, 1 semaine.
- Cornell University, Sept. 02, 2 semaines.
- Brown University, Avr. 98, 1 semaine.

Chapitre 4

Publications

Documents universitaires

1. J-C. Régin : *"Modélisation et Contraintes Globales en Programmation par Contraintes"*, Habilitation à diriger des recherches, Université de Nice-Sophia Antipolis, Novembre 2004.
2. J-C. Régin : "*Développement d'outils algorithmiques pour l'Intelligence Artificielle. Application à la chimie organique*", Thèse de Doctorat, Université Montpellier II, Décembre 1995.
3. J-C. Régin : "*Apprentissage des liaisons stratégiques en chimie organique*", Rapport de DEA, Université Montpellier II, 1990.

Chapitre de Livre

1. J-C. Régin : "Global Constraints" in CPAIOR Edited Collection, M. Milano and P. Van Hentenryck editors, 2009 **to appear**.
2. P. Schaus, Y. Deville, P. Dupont and J-C. Régin : "Simplification and extension of the SPREAD Constraint", in "Future and Trends of Constraint Programming", p.95-99, 2007
3. J-C. Régin : "Global Constraints and Filtering Algorithms", in "Constraints and Integer Programming Combined", Kluwer, M. Milano editor, 2003.

Actes de Conférence

1. J-C. Régin and M. Rueher (Eds.) "Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems", First International Conference, CP-AI-OR'04, Nice, France, April 20-22, 2004, Proceedings Series : Lecture Notes in Computer Science , Vol. 3011.

Revues

1. J-C. Régin, M. Rueher : "Inequality-sum : a global constraint capturing the objective function", **RAIRO Operations Research**, 39, pp 123-139, 2005.
2. C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang : "An Optimal Coarse-grained Arc Consistency Algorithm", **Artificial Intelligence**, vol 165 (2), pp 165–185, 2005.

3. J-C Régin : “Cost based Arc Consistency for Global Cardinality Constraints”, **Constraints**, an International Journal, pp 387–405, Vol 7, Issue 3-4, 2002.
4. J-C. Régin : “Minimization of the number of breaks in sports scheduling problems using constraint programming”, **DIMACS Series in Discrete Mathematics and Theoretical Computer Science**, Volume 57, pp 115–130, 2001.
5. C. Bessière, E.C Freuder and J-C. Régin : “Using Constraint Metaknowledge to Reduce Arc Consistency Computation”, **Artificial Intelligence**, vol.107 (1), pp 125–148, 1999.

Conférences internationales avec comité de lecture

Conférences très sélectives

Le taux d'acceptation des conférences CP, CP-AI-OR, IJCAI et AAAI se situe autour de 30%. Le sélection se fait à partir d'articles et non de résumés. Une version finale tenant compte des remarques des relecteurs est exigée, et est publiée par un éditeur de renom (essentiellement Kluwer ou Elsevier). J'ai publié 15 articles à CP, 4 à CP-AI-OR, 4 à l'IJCAI et 4 à AAAI dont voici le détail :

1. J-C. Régin : "Simpler and incremental consistency checking and arc consistency filtering algorithms for the weighted spanning tree constraint", **CP-AI-OR'08**, Paris, France, 2008.
2. P. Schaus, Y. Deville, P. Dupont and J-C. Régin : "The Deviation Constraint", **CP-AI-OR'07**, Brussels, Belgium, pp. 260-274, 2007.
3. W-J. van Hoeve and J-C. Régin : "Open Constraints in a Closed World", **CP-AI-OR'06**, Cork, Ireland, pp. 244-257, 2006.
4. J-C. Régin : "Maintaining arc consistency algorithms during the search without additional space cost", **CP'05**, Sitges, Spain, 2005.
5. J-C. Régin : "AC-* : A Configurable, Generic and Adaptive Arc Consistency Algorithm", **CP'05**, Sitges, Spain, 2005.
6. G. Pesant and J-C. Régin : "SPREAD : A Balancing Constraint Based on Statistics", **CP'05**, Sitges, Spain, 2005.
7. O. Lhomme and J-C. Régin : "A Fast Arc Consistency Algorithm for n-ary Constraints", **AAAI-05**, Pittsburgh, USA, 2005.
8. J-C. Régin : "Combination of Among and Cardinality Constraints", **CP-AI-OR'05**, Prague, Czech Republic, 2005.
9. J-C. Régin and C. Gomes : "Cardinality Matrix Constraint", **CP'04**, Toronto , Canada, pp 572–587, 2004.
10. J-C. Régin : "Using Constraint Programming to solve the Maximum Clique Problem", **CP'03**, Kinsale, Ireland, pp 634–648, 2003.
11. T. Petit, J-C. Régin, and C. Bessière : “Range-based Algorithm for Max-CSP”, **CP'02**, Ithaca, NY, USA, pp 280–294, 2002.
12. C. Le Pape, L. Perron, J-C. Régin, and P. Shaw : “Robust and Parallel Solving of a Network Design Problem”, **CP'02**, Ithaca, NY, USA, pp 633–648, 2002.
13. J-C. Régin, T. Petit, C. Bessière, and J-F. Puget : “New Lower Bounds of Constraint Violations for Over-Constrained Problems”, **CP'01**, Chyprus, pp 332–345, 2001.
14. T. Petit, J-C. Régin, and C. Bessière, "Specific Filtering Algorithms for Over-Constrained Problems",**CP'01**, Chyprus, pp 451–463, 2001.

15. C. Bessière and J-C. Régin, “Refining the Basic Constraint Propagation Algorithm”, **IJCAI-01**, Seattle, WA, USA, pp 309–315, 2001.
16. J-C Régin and M. Rueher, "A global constraint combining a sum constraint and binary inequalities", **CP'00**, Singapore, pp 384–395, 2000.
17. J-C Régin, T. Petit, C. Bessière, and J-F Puget : “An Original Constraint Based Approach for Solving Over Constrained Problems”, **CP'00**, Singapore, Singapore, pp 543–548, 2000.
18. J-C Régin : “Arc Consistency for Global Cardinality Constraints with costs”, **CP'99**, Alexandria, VA, USA, pp 390–404, 1999.
19. C. Bessière and J-C Régin : “Enforcing arc consistency on global constraints by solving subproblems on the fly”,**CP'99**, Alexandria, VA, USA, pp 103–117, 1999.
20. J-C Régin : “The Symmetric Alldiff Constraint”, **IJCAI-99**, Stockholm, Sweden, pp 420–425, 1999.
21. J-C. Régin and J-F. Puget : “A filtering algorithm for global sequencing constraints”, **CP'97**, Austria, pp 32–46, 1997.
22. C. Bessière and J-C. Régin : “Arc consistency for general constraint networks : preliminary results”, **IJCAI-97**, Nagoya, Japan, pp 398–404, 1997.
23. C. Bessière and J-C. Régin : “MAC and Combined Heuristics : Two Reasons to Forsake FC (and CBJ ?) on Hard Problems”, **CP'96**, Cambridge, MA, USA, pp 61–75, 1996.
24. J-C. Régin : “Generalized Arc Consistency for Global Cardinality Constraint”, **AAAI-96**, Portland, OR, USA, pp 209–215, 1996.
25. T. Schiex, J-C. Régin, C. Gaspin and G. Verfaillie : “Lazy Arc Consistency”, **AAAI-96**, Portland, OR, USA, pp 216–221, 1996.
26. C. Bessière, E.C Freuder and J-C. Régin : “Using Inference to Reduce Arc Consistency Computation”, **IJCAI-95**, Montréal, Canada, pp 592–598, 1995.
27. J-C. Régin : “A filtering algorithm for constraints of difference in CSPs”, **AAAI-94**, Seattle, WA, USA, pp 362–367, 1994.

Article Invité

1. P. Van Hentenryck, L. Michel, L. Perron, and J-C Régin, "Constraint Programming in OPL", **PPDP'99**, Paris, France, invited paper, pp 98–116, 1999.

Autres conférences sélectives

1. T. Petit, J-C Régin, and C. Bessière, "Meta-Constraints on violations for over-constrained problems", **ICTAI-2000**, Vancouver, Canada, pp 358–365, 2000.
2. C. Bessière and J-C. Régin : “Using bidirectionnality to speed-up arc-consistency processing”, Constraint Processing, **Lecture Notes in Computer Science**, M. Meyer ed., Springer-Verlag, 923, 1995, pp 157–170.
3. C. Bessière and J-C. Régin : “An arc-consistency algorithm optimal in the number of constraint checks”, **ICTAI'94**, New Orleans, USA, pp 397–403, 1994.
4. J-C. Régin, O. Gascuel and C. Laurenço : “Machine Learning of strategical knowledge in organic synthesis from reaction databases”, Selected papers from the First European Conference on Computational Chemistry, Conference Proceedings 330, American Institute of Physics, Nancy, 1994, pp 618–623.

5. P. Vismara, J-C. Régin, J. Quinqueton, M. Py, C. Laurenço, and L.Lapied : "RESYN : Un système d'aide à la conception de plan de synthèse en chimie organique", **Avignon-92**, Les systèmes experts et leurs applications, 12ème Journées Internationales, Avignon, France, pp 305–318, 1992.
6. A. Escousse, C. Sgro, M. Biour, J-C. Régin, and V. Rigoulot : "Early Detection of Hepatic Adverse Drug Reactions by the Medical Practitioner : Microcomputerised Bank to Diagnosis", 4^{th} World Conference on Clinical Pharmacology and Therapeutic, Manheim Heidelberg, Germany, 1989.

Workshops internationaux avec comité de lecture

1. P. Schaus, Y. Deville, P. Dupont, J-C. Régin : "Simplification and extension of the SPREAD Constraint", **CP'06**, proceedings **workshop on Constraint Propagation and Implementation**, Nantes, p.72-92, 2006.
2. J-C Régin : "Maintaining arc consistency algorithms during the search with an optimal time and space complexity", **CP'04**, proceedings **workshop on Constraint Propagation and Implementation**, Toronto, Canada, 2004.
3. J-C Régin : "CAC : A configurable, generic and adaptive arc consistency algorithm", **CP'04**, proceedings **workshop on CP and implementation**, Toronto, Canada, 2004.
4. T. Petit, C. Bessière, and J-C Régin : "A General Conflict-Set Based Framework for Partial Constraint Satisfaction", **CP'03**, proceedings **workshop on Soft Constraints**, Kinsale, Ireland, 2003.
5. J-C. Régin : "Solving the Maximum Clique Problem with Constraint Programming", **CP-AI-OR'03**, Montreal, Canada, 2003.
6. T. Petit, J-C. Régin, and C. Bessière : "Range-based Algorithm for Max-CSP", **ECAI-2002**, proceedings **workshop on Modelling and Solving Problems with Constraints**, Lyon, France, 2002.
7. J-C. Régin and M. Rueher : "A global constraint combining a sum constraint and binary inequalities", **IJCAI-99**, proceedings **Workshop on Non Binary Constraints**, Stockholm, Sweden, pp F :1–13, 1999.
8. J-C. Régin : "Minimization of the number of breaks in sports scheduling problems using constraint programming", proceedings **DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimization**, pp P7 :1–23, 1998.
9. C. Bessière and J-C. Régin : "Local Consistency on Conjunctions of Constraints", **ECAI-98**, proceedings **Workshop on Non Binary Constraints**, Brighton, England, pp 53–60, 1998.
10. C. Gaspin and J-C. Régin : "Application of maximal constraint satisfaction problems to RNA", **CP'97**, proceedings **Workshop in Bioinformatics**, Austria, 1997.
11. C. Bessière and J-C. Régin : "An arc-consistency algorithm optimal in the number of constraint checks", **ECAI'94**, proceedings **Workshop on Constraint Processing**, Amsterdam, The Netherlands, pp 9–16, 1994.

Conférences nationales avec comité de lecture

1. P. Schaus, Y. Deville, P. Dupont and J-C. Régin : "La Contrainte Déviation", **JFPC'07**, Rocquencourt, France, p.173-182, 2007.

2. J-C. Régin : "CAC : Un algorithme d'arc-consistance configurable, générique et adaptatif", **JNPC'04**, Angers, France, 2004.
3. T. Petit, C. Bessière, and J-C Régin : "Détection de Conflits pour la Résolution de Problèmes Sur-contraints", **JNPC'03**, Amiens, France, pp 293–308, 2003.
4. R. Bernhard, J. Chambon, C. Le Pape, L. Perron, and J-C. Régin : "Résolution d'un problème de conception de réseau avec Parallel Solver", **JFPLC'2002**, Nice, France, pp 151–166, 2002.
5. J-C. Régin, J-F. Puget, and T. Petit : "Representation of soft constraints by hard constraints", **JFPLC'2002**, Nice, France, pp 191–198, 2002.
6. T. Petit, J-C. Régin, and C. Bessière, "Algorithmes de filtrage specifiques pour les problemes sur-contraints",**JNPC'2001**, Toulouse, France, pp 233–246, 2001.
7. C. Bessière and J-C. Régin, "Refining the Basic Constraint Propagation Algorithm", **JFPLC 2001**, Paris, France, 2001
8. C. Fagot and J-C. Régin : "CoNNei : une méthode conceptuelle de voisinage", **JFA-96**, Journées Françaises de l'Apprentissage, Sète, France, pp 330-333, 1996.

Communications invitées

1. J-C. Régin, "How to prevent tall trees from growing to the sky", invited talk, **CSCLP 2007 : Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming**, Rocquencourt, France, 2007.
2. J-C. Régin, "Alldifferent and Cardinality Constraints", **Second International Summer School on Constraint Programming**, Samos, Greece, July 2006.
3. J-C. Régin, "Global Constraints", invited tutorial, **First International Summer School on Constraint Programming**, Aquafredda di Maratea, Italy, Sept 2005.
4. J-C. Régin, "Implementation of Arc Consistency Algorithms in a Solver", invited talk, **Constraint Propagation and Implementation workshop**, CP'04, Toronto, Canada, Sept 2004.
5. J-C. Régin, "Graph Theory and Constraint Programming", invited tutorial, Master Class, **CP-AI-OR'04**, Nice, France, Avril 2004.
6. N. Beldiceanu and J-C. Régin, "Global Constraints", invited tutorial, **CP'02**, Ithaca, USA, Sept 2002.
7. J-C. Régin, "Implementation of Soft Constraints", invited talk, **TRICS workshop**, CP'02, Ithaca, USA, Sept 2002.
8. J-C. Régin, "Global Constraints", invited talk, **CP-AI-OR'02**, Le Croisic, France, May 2002.
9. J-C. Régin and B. Smith, "Modelling and Algorithmic Techniques in Constraint Programming", invited tutorial, **Dagsthul seminar on Constraint programming and Integer programming**, 16–21 Jan 2000.

Séminaires invités

1. J-C. Régin : "General Principles of Constraint Programming", CSE Colloquium Distinguished Speaker, **Univ. Nebraska-Lincoln**, Lincoln, USA, April 2008.

2. J-C. Régin : "Current Challenges in Constraint Programming", Département d'Ingénierie Informatique, **Univ. Catholique de Louvain-la-Neuve**, Belgium, Nov. 2007
3. J-C. Régin : "General Principles of Constraint Programming", Post-Graduate School of Engineering of the **Federal University of Rio de Janeiro**, Brazil, Aug 2007
4. J-C. Régin : "Modelling Problems in Constraint Programming", **Cork Constraint Computation Center**, Cork, Ireland, Aug 2005
5. J-C. Régin : "Using Constraint Programming to Solve the Maximum Clique Problem", Computer Science Seminar, **Brown University**, Providence, USA, May 2005
6. J-C. Régin : "General Principles of Constraint Programming", IISI Seminar, **Cornell University**, Ithaca, USA, Feb 2005
7. J-C. Régin : "Introduction à la Programmation par Contraintes", Séminaire RO, **Université de Montreal**, Montréal, Canada, May 2003
8. J-C. Régin : "Principles of Constraint Programming", private worshop organized by G. Nemhauser, **Georgia Tech University**, Atlanta, USA, May 1999.

Tutorial

1. J-C. Régin, "Modeling Problems in Constraint Programming", **CP'04**, Toronto, Canada, Sept 2004.

Communications Orales

1. J-C. Régin, "An Efficient Constraint to solve Car Sequencing Problems with Constraint Programming", **Cors/Informs 2004**, International Meeting, Banff, Canada, May 2004. Invited by A. Lody.
2. J-C. Régin, "Using Constraint Programming to solve the Maximum Clique Problem", **Cors/Informs 2004**, International Meeting, Banff, Canada, May 2004. Invited by P. Van Hentenryck.
3. J-C. Régin, "An original method to deal with distance constraints", **Cors/Informs 2004**, International Meeting, Banff, Canada, May 2004. Invited by J. Hooker.
4. J-C. Régin, "Constraint Programming and Sports League Scheduling", **Cors/Informs 2004**, International Meeting, Banff, Canada, May 2004. Invited by M. Trick.
5. C. Gomes, J-C. Régin, "Modelling Alldiff matrix models in Constraint Programming", **Optimization days**, Montreal, Canada, May 2003.
6. J-C. Régin, "Combination of Cardinality and Sequence Constraints", **Optimization days**, Montreal, Canada, May 2003.
7. T. Petit, J-C. Régin, and C. Bessière, "Generalization of constructive disjunction for over-constrained problems", **Informs**, Miami, Floride, Nov. 2001. Invited by G. Pesant.
8. T. Petit and J-C. Régin, "An original constraint based approach for solving over constrained problems", **ISMP**, Aug 2000, Atlanta.
9. J-C. Régin and J-F. Puget, "Solving Car Sequencing Problems with Constraint Programming", **ISMP**, Aug 2000, Atlanta. Invited by M. Junger.

10. J-C. Régin, "Constraint Programming and Sports Scheduling Problems", **Informs**, May 1999, Cincinnati. Invited by M. Trick.
11. J-C. Régin, "Flow Theory and Constraint Programming", **Informs**, May 1999, Cincinnati. Invited by K. McAloon.
12. J-C. Régin, "Modeling and Solving Sports League Scheduling with Constraint Programming", **Informs**, April 1998, Montreal. Invited by K. McAloon.
13. J-C. Régin, "Intérêt de la théorie des flots en programmation par contraintes", **1er congrès ROADEF**, Paris, Jan. 1998. Invited by Y. Caseau.

Chapitre 5

Charges collectives

Tout d'abord, j'ai déjà mentionné dans ce dossier diverses activités collectives ayant trait à l'animation de la communauté scientifique dont j'ai eu la charge, notamment l'organisation et la présidence du comité de programme d'une conférence internationale (cf section 3.6).

Lorsque j'étais directeur de la programmation par contraintes à ILOG, je dirigeais et animais une équipe de sept personnes dont un doctorant, quatre titulaires d'un doctorat, les autres étant ingénieurs. Trois personnes étaient à ILOG Gentilly, trois à ILOG Sophia Antipolis et une à ILOG Madrid. La composition de cette équipe a évolué dans le temps.

L'animation et la gestion d'une équipe à ILOG consiste à s'assurer que les clients obtiennent des réponses de la part des développeurs lorsque cela est demandé (c'est la priorité), à définir des objectifs avec chacun des développeurs ou chefs de projets, à s'assurer que les procédures de qualité sont respectées ainsi que les procédures d'écriture du code. Il s'agit aussi de contrôler les relations entre les membres de l'équipe ainsi que celles avec les autres équipes. Il faut également de permettre à chacun de faire carrière et à faire les propositions d'augmentations.

Par ailleurs, en tant que directeur, je devais collaborer avec divers services de la société comme : le marketing pour la définition de nouvelles fonctionnalités et la promotion des produits, la documentation, la hot-line, le service qualité, la production. Enfin, ILOG Solver est un produit qui est à la base de nombreux autres, appelés add-ons. Il est important de s'assurer que la communication avec les add-ons se passe bien.

Chapitre 6

Transferts Technologiques

Je pourrais utiliser mon expérience acquise à ILOG à la fois au niveau enseignement et pour ceux qui voudraient transférer leurs travaux dans le monde industriel. Pour montrer cela, j'ai décidé de consacrer quelques lignes de ce dossier aux transferts technologiques.

6.1 Gestion de Librairie logicielle

ILOG produit des moteurs d'optimisation fournis sous la forme de librairies logicielles. ILOG ne développe pas de solutions dédiées, mais aide parfois au développement des applications. Le développement d'un produit en vue de sa commercialisation et les réalités du monde industriel imposent un certain nombre de contraintes. Par exemple, de nombreux utilisateurs ne lisent pratiquement pas le manuel de l'utilisateur et se contentent de faire du mimétisme a partir de code existant et de parcourir le manuel de référence.

Il n'existe pas d'architecture standard pour le développement de logiciels de haute technologie. Néanmoins, après plusieurs années d'expérience, je me suis aperçu qu'un certain nombre de principes de base du génie logiciel devait être gardé à l'esprit lors de la définition d'une nouvelle fonctionnalité ou l'écriture d'un nouveau code. Je résume ici les principes les plus importants à mon avis. De plus amples détails peuvent être trouvés dans la littérature concernant le génie logiciel, notamment dans le fameux livre de E. Raymond : "The cathedral and the bazaard".

• **Simplicité des concepts et du code**. Un produit doit être simple à utiliser lorsque l'on veut faire des choses simples et peut être plus complexe à utiliser pour faire des choses plus compliquées. Mais, un code complexe doit s'obtenir par combinaison de principes simples. Des fonctions dédiées, simples et précises, donc maîtrisables, communiquant entre elles, sont souvent plus pratiques que des fonctions à vocation générale dont le comportement dépend de nombreux paramètres. En effet, le comportement d'une fonction ne doit jamais surprendre l'utilisateur. Une fonction qui correspond à un comportement bien identifiable permet à l'utilisateur un contrôle fin et précis du moteur.

• **Robustesse**. Les données sont le plus souvent extraites de bases de données et les modèles sont de plus en plus souvent engendrer automatiquement. Aussi les fonctions sont fréquemment employées avec des paramètres incohérents. On devra donc, entre autres, tester si les contraintes ne sont pas trivialement satisfaites ou violées. C'est par exemple le cas d'une contrainte alldiff impliquant deux fois la même variable. Par ailleurs, il faut assurer la compatibilité entre toutes les versions (majeures ou mineures), afin de permettre à l'utilisateur une migration plus aisée

vers les nouvelles versions et éviter les problèmes de "backport", c'est-à-dire de modifications des versions antérieures. La toute dernière version corrigeant un bug pourra ainsi toujours être utilisée par un utilisateur quelque soit la version avec laquelle il a développé son application.

• **Evolutivité**. Le code doit pouvoir s'adapter aux demandes futures qui ne sont pas prévisibles. Or, les ingénieurs ont été habitué à réfléchir sur des problèmes bien définis. Dans un énonce de mathématiques tel qu'on en rencontre à l'école, le monde est bien carré et bien défini et on ne se pose pas de question sur son évolution. Dans la vie réelle c'est bien différent. Il vaut mieux étudier le problème et la solution proposée pendant plusieurs jours pour s'assurer qu'une évolution sera possible. Il est donc préférable d'éviter de faire trop de nouvelles fonctionnalités et penser aux vrais problèmes actuels. Aussi, une solution basée sur la combinaison de modules est bien souvent meilleure qu'une solution très générale. De toute façon le code sera toujours utilisé de manières différentes de celles auxquelles on a pensées. En conséquence, la disponibilité rapide d'un code utile et améliorable est préférable à l'attente d'un code soi-disant parfait.

• **Réutilisabilité**. Il est souvent plus intéressant de faire un algorithme simple avec des structures de données avancées et réutilisables, qu'un algorithme complexe avec des structures de données simples. En effet, dans le premier cas on peut se concentrer indépendamment sur chacune des parties à optimiser. De plus, toute amélioration d'une structure de données conduira à l'amélioration de toutes les fonctions l'utilisant, et donc de plusieurs parties du produit.

• **Performance**. L'amélioration des performances doit être ciblée. Le code n'est pas critique partout. La modularité et la réutilisabilité sont souvent les meilleurs moyens de garder de bonnes performances pendant l'évolution du produit. L'écriture d'un nouveau code plus dédié ou l'amélioration du mécanisme de communication sont deux des méthodes les plus sûres pour permettre une amélioration des performances sur le long terme. Il est également inutile de sur-optimiser les premières versions, car c'est à l'usage que l'on pourra mieux comprendre les parties critiques du code. Enfin, la simplicité est aussi un critère lié à la performance. Par exemple lors de l'écriture de tableaux extensibles, la solution qui consiste à créer un nouveau tableau deux fois plus grand que le précédent est certainement la meilleure. On ne perd qu'au plus un facteur 2 en mémoire, mais on gagne un code très simple et court, donc rapide.

6.2 Intégration de travaux de recherche dans un produit industriel

Un moteur de PPC industriel est un outil très général. Il est donc difficile d'intégrer d'autres méthodes à vocation générale. En effet, il n'est pas question de remettre en cause le fonctionnement du moteur afin d'y intégrer une méthode qui n'est efficace que pour certains problèmes uniquement. Il n'est pas acceptable d'améliorer les performances des applications de certains utilisateurs au détriment d'autres. En outre, il faut être conscient que la multiplication de méthodes entraîne un sur-coût industriel non négligeable (apprentissage de la méthode, maintenance, évolution...). Bien souvent, les solutions proposées dans la littérature sont complètes : un problème donné est résolu grâce à l'utilisation de certaines contraintes et d'une stratégie de sélection de variables et de valeurs très particulières. Les stratégies se combinant plutôt difficilement, la modularité en PPC est donc principalement obtenue en combinant des contraintes. Aussi, il est nécessaire d'extraire une partie de la connaissance utilisée dans les stratégies pour définir de nouvelles contraintes, qui elles pourront être utilisées dans n'importe quel type d'application et ce indépendamment des stratégies

de sélection de variables ou de valeurs choisies.

Par ailleurs, certaines études s'avèrent irréalistes dans le cas général. Nous pouvons citer comme exemple les CSP valués qui proposent d'associer à chaque combinaison de valeurs un coût de violation de la contrainte pour cette combinaison. Cette méthode entraîne bien évidemment un coût mémoire exponentiel et n'est donc pas facilement intégrable dans un moteur à vocation générale. Nous donnons dans le seconde partie de ce mémoire une méthode beaucoup plus réaliste et montrons comment les travaux existants peuvent être intégrés grâce à cette méthode.

En revanche, il est aisé d'introduire des algorithmes indépendants comme les filtrages associés aux contraintes. Dès lors, tout algorithme de filtrage publié est systématiquement étudié et éventuellement incorporé dans le produit. Cela peut aussi être le cas de certaines méthodes proposant de casser les symétries du problème initial. Ces différents algorithmes doivent néanmoins être adaptés afin d'être utilisable pour n'importe quel type d'application, ce qui n'est pas nécessairement le souci de l'auteur de l'idée.

Chapitre 7

Lettres de Recommandation

Vous trouverez dans les pages qui suivent les lettres :
– de Michel Rueher, Professeur, Univ. Nice-Sophia Antipolis
– de M. Philippe Baptiste, CNRS, Ecole Polytechnique
Conformément à l'usage en vigueur aux États-Unis, la lettre de recommandation du Professeur Pascal Van Hentenryck de l'Université de Brown, vous sera directement envoyée par ses soins.

Philippe Baptiste
Ecole Polytechnique, LIX
F-91128 Palaiseau

Sydney, December 5, 2008

Lettre de recommandation

Je connais Jean-Charles Régin depuis 1995 et j'ai eu le plaisir de collaborer avec lui tant sur des projets industriels qu'académiques.

M. Régin était jusqu'à très récemment l'un des directeur R&D d'Ilog, une société spécialisée dans l'édition de logiciels. Il était l'architecte d'Ilog Solver, le plus puissant outil actuel de résolution de contraintes. Les travaux de M. Régin ont permis un rapprochement significatif des communautés scientifiques de Recherche Opérationnelle et de Programmation Logique. Il a en particulier montré comment utiliser des résultats de la théorie des graphes au sein des outils de contraintes et a ainsi contribué à ouvrir un nouveau domaine de recherche. Ses travaux sont très largement cités dans les publications scientifiques internationales et il joue déjà un rôle majeur dans l'animation de la communauté de programmation par contraintes.

En plus de ses activités industrielles et de ses activités de recherches, Jean-Charles Régin a souvent montré son intérêt pour l'enseignement et pour la diffusion de la connaissance scientifique auprès d'étudiants. Il intervient ainsi tous les ans, avec succès, dans l'un de mes cours de M1 à l'Ecole Polytechnique. J'ai aussi assisté à plusieurs de ses " tutoriaux " dans des conférences et j'ai été frappé par la qualité pédagogique de ses interventions.

Pour conclure, je recommande sans aucune réserve Jean-Charles Régin. Il fera un excellent Professeur des Universités, et son laboratoire d'accueil bénéficiera du rayonnement de ses travaux et des collaborations industrielles qu'il ne manquera pas d'apporter.

Philippe Baptiste
Director of the Ecole Polytechnique CS
Lab (LIX)

Sophia-Antipolis, le 11 décembre 2008

Je tiens tout d'abord à attirer l'attention de mes collègues du CNU sur la qualité du dossier que nous présente aujourd'hui Monsieur Jean-Charles Régin et sur l'intérêt que présente son recrutement pour notre communauté scientifique. Le dossier de Monsieur Jean-Charles Régin est un dossier original et parmi les tous meilleurs qu'on puisse trouver actuellement dans ce domaine scientifique.

Monsieur Jean-Charles Régin a soutenu en 1995 une thèse remarquable dont les résultats sont encore fréquemment cités aujourd'hui. Monsieur Jean-Charles Régin a ensuite développé des travaux fondateurs sur les contraintes globales qui ont eu de très nombreuses applications et ont ouvert de nouvelles voies de recherche. De ce fait, Monsieur Jean-Charles Régin a un rayonnement indiscutable dans notre communauté scientifique et est fréquemment invité à présenter ses résultats dans les meilleures universités étrangères. Un chercheur aussi prestigieux que Monsieur Pascal Van Hentenryck indique dans son rapport d'HDR qu'il s'agit certainement de la meilleure HDR qu'il a vu ces dix dernières années.

L'originalité du dossier de Monsieur Jean-Charles Régin réside aussi dans sa une solide expérience industrielle : il a eu un rôle clé dans les développements de ILOG Solver, un logiciel complexe diffusé à grande échelle et dans des environnements variés. Je tiens à souligner ici qu'il a participé à toutes les étapes du développement, de la maintenance et de la diffusion de ce logiciel. Une expérience précieuse et rare dans le corps enseignant universitaire.

Le recrutement de Monsieur Jean-Charles Régin comme Professeur des Universités ne pourra être que très bénéfique pour l'ensemble de notre communauté académique. C'est pourquoi nous l'avons embauché cette année sur un poste de Professeur associé et j'espère naturellement que nous arriverons à pérenniser sa position. Je ne peux donc qu'apporter un soutien inconditionnel et sans réserve au renouvellement de sa qualification aux fonctions de Professeur des Universités.

Michel Rueher, Professeur

École polytechnique de l'université de Nice – Sophia Antipolis Département Sciences Informatiques
930, routes des Colles – BP 145 – 06903 Sophia Antipolis Cedex
Tél. : 33 (0)4 92 96 51 54 - Fax : 33 (0)4 92 96 50 55 - email : rueher@essi.fr

Chapitre 8

Rapports de Présoutenance

8.1 Rapports d'HDR

Vous trouverez dans les pages qui suivent les rapports :
- du Pr. Pascal Van Hentenryck
- du Pr. Eugene Freuder
- du Pr. Jacques Carlier

Ainsi que le rapport de la soutenance.

BROWN

Department of Computer Science
http://www.cs.brown.edu/

Pascal Van Hentenryck
Brown University
Department of Computer Science
Box 1910, Providence, RI 02912
Email: pvh@cs.brown.edu

October 22, 2004

Professor Michel Rueher
ESSI (Ecole Suprieure en Sciences Informatiques),
Route des colles, BP 145,
06903 Sophia Antipolis,
France

Dear Michel,

I am responding to your letter regarding the habilitation report of Dr. J.-C. Régin. It is with great pleasure that I am writing this report. I have been following Dr. Régin's research for about 10 years, witnessing his significant impact on the field of constraint programming and combinatorial optimization.

Constraint programming is now largely recognized as one of the fundamental tools for combinatorial optimization, both academically and industrially. It is widely used in industry to solve complex resource-allocation and scheduling problems. The field also has its own successful conferences, its journal, and it is typically represented by multiple sessions at major conferences on operations research.

Dr. Régin has contributed to these successes in significant ways. He has been one of the pioneers of the concept of global constraints and has published many algorithms for global constraints that are embedded in most constraint programming tools. Equally important, Dr. Régin has convincingly demonstrated how global constraints are fundamental tools for solving complex applications in sport scheduling and other resource allocation problems. Many industrial solutions to sport scheduling problems now use constraint programming, which is largely recognized as superior to integer programming for this class of applications. Dr. Régin has also made significant contributions to many other topics, including the implementation of constraint

propagation algorithms and filtering algorithms for soft constraints. And, finally, his scientific work shows great taste, always reaching for clean and elegant solutions.

Let me now go over some of Dr. Régin's key constributions in more detail. Perhaps his most seminal contributions are a series of publications on global constraints: alldifferent, global cardinality, and global cardinality with costs. The beauty in these papers is twofold. On the one hand, these constraints are fundamental modeling tools for a variety of applications. On the other hand, the paper presents filtering algorithms enforcing arc consistency which is, in some sense, optimal from the standpoint of reducing the search space. These papers set the standard for research on global constraints in terms of modeling power, pruning, and efficiency. Of course, Dr. Régin also contributed many other global constraints including some for car sequencing and maximum clique. (Note that N. Beldiceanu is also a major contributor to this area. He coined the term "global constraint" and his impact has also been significant, albeit in different ways.)

Perhaps the second most significant research results of Dr. Régin are his application papers and presentations about sport scheduling and, more recently, on finding maximum cliques. These papers, in particular those on sport scheduling, have clearly identified an area where constraint programming excelled both as a modeling and as an implementation technology. These applications, together with results on scheduling, have helped constraint programming increase its visibility in the operations research community.

The third class of significant results are Dr. Régin's papers on generic filtering algorithms, including the basic propagation algorithm (IJCAI'01), generalized arc consistency (CP'99) and, more recently, the paper unifying the various algorithms in a unique framework. What is remarkable here is that some of these papers include new insights on topics that seem to have been around forever.

Finally, last but not least, the research of Dr. Régin on soft constraints is a promising direction to approach a class of problems which are fundamental in practice. New results by Dr. Régin and other researchers, started to appear in this direction and may have significant impact in coming years.

This brief summary should give you an idea of the scope and depth of Dr. Régin's contributions to the area. It does not capture his impact through his talks, his

tutorials (e.g., in CP'02 and CP'04), his organization of conferences and workshops (including CP'AI'OR'04 with Professor Rueher), his influence in driving the field in some directions, and his industrial work in developing and maintaining high-quality software. Dr. Régin also supervised an excellent PhD student (T. Petit) and is co-advising a student currently with Professor Minoux.

In conclusion, it should be very clear that Dr. Régin has been one of the most significant contributors to the area of constraint programming in the last decade, both in terms of technical results and of impact on the field. Dr. Régin, as far as I can judge, has largely exceeded all the traditional expectations for an "habilitation à diriger des recherches" and PhD students will be fortunate to benefit from his expertise and vision in the years to come. From all the habilitations I have taken part of in the last decade, this is probably the strongest one. Please do not hesitate to contact me if you need additional detail.

Best Regards,

Pascal Van Hentenryck
Professor of Computer Science

Review of HDR for Dr. Jean-Charles Régin

Eugene C. Freuder
SFI Research Professor
Director, Cork Constraint Computation Centre
University College Cork
Cork, Ireland

I heartily recommend Jean-Charles Régin for his HDR. Unfortunately I could not read his French document, but I base my assessment on his English papers, his record, and my personal knowledge.

Dr. Régin is one of the brightest young scientists in the field of constraint programming. He is making important contributions to the field. He not only excels at developing new algorithms and studying them experimentally, he also sees the "big picture". He understands the context and the larger issues and implications of his work. I always find discussions with Dr. Régin to be very stimulating.

His publication record is very strong. He does not have as many journal publications as one might expect, perhaps because he is in an industry lab and not in academia; but his record of publications in the most selective conferences in our field, which are virtually archival, is outstanding for its quantity, quality and consistency over an extended period of time.

Dr. Régin is one of the pioneers and leading experts in the world on several topics in constraint programming:

- Filtering Algorithms

He has developed interesting new approaches to the central arc consistency inference method in constraint programming. He has extended arc consistency beyond discrete, finite domain, binary constraints, and developing specialized filtering algorithms for global constraints.

- Global Constraints

He has identified important global constraints and developed efficient filtering algorithms to exploit them. In particular, he was one of the first to exploit the very popular all-diff constraint, developed a filtering algorithm for global sequencing constraints and a cost based arc consistency algorithm for the global cardinality constraint, and introduced the inequality-sum global constraint. He identified subproblem solution as a method of distinguishing useful global constraints, and shown how Max-CSP can be used as a global constraint to isolate over-constrained parts of a problem.

- Modelling

He has recognized the crucial importance of modelling and helped inspire work in this field by suggesting general lessons to be learned from insightful and successful case studies.

Dr. Régin has successfully applied his methods to interesting application domains, including sports scheduling and nework design. Again, he sees the big picture: the former serves as an excellent case study in the importance and methods of modelling; from the latter he draws lessons about robustness. For the much-studied maximum clique problem he is able to solve a number of instances for the first time.

Of course, Dr. Régin has had co-authors for some of this work (I myself have been a co-author), but he has also published alone, and is clearly capable both of independent work, and of contributing to a fruitful collaboration.

Once again, I strongly recommend Dr. Régin for his HDR.

Rapport sur le dossier d'Habilitation à Diriger des Recherches de Jean-Charles Régin sur « Modélisation et Contraintes Globales en Programmation par Contraintes ».

Préambule

Le document fourni aux rapporteurs inclut le CV de Jean-Charles Régin, une description détaillée de ses activités de recherche et une sélection de dix de ses articles. Je vais ci-dessous présenter brièvement le candidat, avant d'analyser les points forts de ses recherches, qui s'effectuent en Programmation par Contraintes (PPC), autour des algorithmes de filtrage et du traitement d'applications. Cela me permettra de donner un avis motivé sur sa demande d'Habilitation à Diriger des Recherches.

Le candidat

Jean-Charles Régin a 38 ans. Il est depuis trois ans Directeur de la Programmation par Contraintes chez ILOG, et a soutenu en décembre 1995 un doctorat sur « Développements d'Outils Algorithmiques pour l'Intelligence Artificielle. Applications à la Chimie Organique ». Son doctorat a été encadré par Olivier Gascuel au LIRMM de l'Université de Montpellier II, et financé par SANOFI CHIMIE et le CNRS. Il s'agissait de concevoir et implémenter RESYN, un système de planification de rétrosynthèse en chimie organique. Dès janvier 1996, le candidat était engagé comme ingénieur-informaticien chez ILOG, où il s'occupe des produits logiciels, ILOG SOLVER et ILOG J SOLVER, qui sont des moteurs de résolution de problèmes d'optimisation. Son travail inclut la gestion de projets nationaux et internationaux, l'encadrement d'ingénieurs et de doctorants, des activités de recherche et de transfert technologique, et la formation continue d'utilisateurs des logiciels. Ses recherches l'ont amené à collaborer avec des collègues de chez ILOG, comme Claude Le Pape et Jean-François Puget, et des chercheurs universitaires comme Christian Bessière, Carla Gomes, Pascal Van Hentenryck, Eugene Freuder, Christine Gaspin, Thomas Schiex et Michel Rueher. Ses résultats scientifiques lui ont permis de publier trois articles dans les revues, Constraints, Discrete Series in Mathematics and Theoretical Computer Science, Artificial Intelligence, un chapitre de livre chez KLUWER, et dix-neuf articles dans des congrès très sélectifs comme CP, IJCAI et AAAI.

Travaux de recherche amont : filtrage et contraintes globales

Une grande partie des recherches de Jean-Charles Régin concerne la conception d'algorithmes pour la PPC. Rappelons qu'en PPC, un problème est modélisé à l'aide de contraintes et de variables, le choix du modèle étant évidemment crucial pour une résolution efficace. 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. La contrainte est binaire si deux variables seulement sont concernées. La PPC utilise pour chaque sous-problème une méthode de filtrage spécifique à ce sous-problème afin de supprimer des valeurs des domaines des variables. La propagation a pour objet de propager les conséquences de ces suppressions. On parle de consistance d'arc d'une contrainte, si chaque valeur du domaine d'une variable est compatible avec la contrainte. Jean-Charles Régin a proposé l'algorithme CAC qui est générique pour la consistance d'arc de contrainte binaire. Il a aussi proposé le schéma général GAC – schéma pour la consistance d'arc de contraintes non binaires. Quand les déductions sont associées à plusieurs contraintes simultanément, on parle de contrainte globale. Le candidat a travaillé sur les contraintes globales en utilisant des méthodes de Recherche Opérationnelle pour concevoir des algorithmes de filtrage efficaces. Ainsi pour la contrainte globale all diff (les valeurs des n variables doivent être distinctes), il s'appuie sur une modélisation par couplage dans un graphe biparti. Plus généralement l'auteur a étudié la contrainte globale de cardinalité qui impose que le nombre de fois qu'une valeur particulière soit prise se situe dans un intervalle. La consistance d'arc est obtenue en recherchant les composantes fortement connexes du graphe d'écart associé à un flot compatible. Une extension prenant en compte les coûts d'utilisation des valeurs est résolue par la recherche de chemins extrémaux du graphe d'écart. Ces méthodes sont également utilisables pour des contraintes k-Diff (au moins k valeurs distinctes), et des contraintes sur des variables ensemblistes. Ce sont des exemples de méthodes proposées par le candidat qui est un spécialiste reconnu de ce domaine très compétitif de recherche en PPC. Six articles sur les dix sélectionnés concernent cet aspect de ses recherches. Ils ont été présentés à JNPC04, CP00, Constraints, CP99, IJCAI99, CP97, IJCAI97 et AAAI 96.

Travaux de recherche aval : Problèmes surcontraints et applications

La PPC n'a d'intérêt que si elle permet de traiter efficacement les applications. En fait, on a souvent besoin d'adapter les outils logiciels au problème traité. Il faut choisir habilement le modèle, mais aussi sélectionner dans la bibliothèque d'algorithmes ceux qui conviennent. C'est pourquoi toute une partie de la recherche de Jean-Charles Régin est orientée vers les applications.

Un problème est surcontraint s'il n'a pas de solution, il faut alors classer les contraintes entre contraintes dures et contraintes molles. Les problèmes surcontraints ont fait l'objet de nombreuses études. Ainsi un modèle bien connu est celui du « Valued-CSP », qui se révèle d'après l'auteur inadapté à la résolution de problèmes réels. C'est pourquoi le candidat a proposé un nouveau modèle, en collaboration avec Thierry Petit, alors étudiant en thèse. Il suggère de remplacer le problème initial par un nouveau problème dans lequel les contraintes dures doivent être satisfaites, alors que les contraintes molles ont un coût de violation. Ce nouveau problème ne doit évidemment pas être surcontraint. On peut aussi minimiser le nombre de contraintes violées à l'aide de méthodes de la littérature de type Branch and Bound, comme PFC-MRDAC, mais cela peut provoquer des dépassements mémoires. Aussi Jean-Charles Régin a-t-il proposé avec Thierry Petit et Christian Bessière des adaptations des

méthodes correspondantes. Il a aussi développé de nouveaux algorithmes de filtrage spécifiques.

Le candidat a travaillé sur de nombreuses applications comme les problèmes du « car sequencing », du « all interval series », du « quasigroup completion » ou encore du « sports sequencing ». Il nous présente dans son document, le traitement de deux applications importantes : la recherche de la taille de la plus grande clique d'un graphe et le dimensionnement d'un réseau de télécommunications. La méthode proposée par Jean-Charles Régin pour la plus grande clique intègre la démarche de Fahle, qui construit une clique de plus en plus grande en filtrant les nœuds restants. Elle utilise également un filtrage basé sur une idée de Bron et Kerbosch, ainsi que de nouvelles propriétés. Une technique de plongée permet de construire rapidement de bonnes solutions. Cette méthode a été testée sur le benchmark d'un challenge DIMACS. Elle est très compétitive face à des méthodes complètes comme celles de Fahle, et pour calculer des bornes inférieures. La méthode développée pour le dimensionnement d'un réseau de télécommunications utilise des variables ensemblistes associées aux chemins dans le réseau. Cela permet de prendre en compte la structure du problème. Des contraintes globales sont implémentées de manière à détecter les nœuds et les arcs devant appartenir à un chemin donné. La méthode a été comparée à un MIP (Mixed Integer Programming). La PPC se révèle particulièrement robuste. Ces travaux sur les applications font l'objet de quatre articles sur les dix sélectionnés par le candidat. L'un est publié dans les DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Les autres ont été présentés aux congrès CP03, CP02 et CP01.

Avis motivé du rapporteur

Jean-Charles Régin a donc travaillé en amont et en aval de la PPC. En amont, il a développé des algorithmes originaux de filtrage pour la consistance d'arc de contraintes locales et globales. En aval, il a pour des applications particulières, proposé des modèles et algorithmes adaptés. Il s'agit d'un travail de recherche de haut niveau comme l'attestent ses nombreuses publications dans des congrès très sélectifs. Jean-Charles Régin est un chercheur confirmé dans son domaine, qui a montré du goût pour l'enseignement, en formation initiale dans le cadre du DEA de Montpellier, et en formation continue dans le cadre de la PPC chez ILOG. Par ailleurs, le candidat a déjà encadré deux doctorants, Thierry Petit et Diego Pons, le premier ayant soutenu en 2002. Pour toutes ces raisons, il mérite amplement d'obtenir l'Habilitation à Diriger des Recherches de l'Université de NICE.

Fait à Compiègne,
Le 15 septembre 2004

Jacques CARLIER
Professeur des Universités
27ème section

UNIVERSITE DE NICE-SOPHIA ANTIPOLIS

ECOLE DOCTORALE DES SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

Rapport de soutenance

D'Habilitation à Diriger les Recherches de Monsieur REGIN Jean-Charles

Titre du mémoire : *Modélisation et contraintes globales en programmation par contraintes*

Membres du jury :

Les travaux de Monsieur Jean-Charles Régin sur la modélisation et les contraintes globales en PPC sont de très haut niveau. Poursuivant des activités de pionnier, il a apporté des contributions exceptionnelles en adaptant des résultats de la théorie des graphes à la PPC et en résolvant des problèmes très difficiles en recherche opérationnelle.

Les connaissances de Monsieur Jean-Charles Régin sont très vastes et couvrent tout le spectre allant de la théorie à la mise en œuvre dans un cadre industriel.

Enfin, le jury a apprécié l'enthousiasme de Monsieur Jean-Charles Régin qui possède d'indéniables qualités pédagogiques.

Signature des membres du jury : **date :**

H. COSNARD Yves CASEAU A. Colmerauer Alain Colmerauer A. RUEHER

F. Fages J. CARLIER

Ecole Doctorale STIC de l'Université de Nice-Sophia Antipolis

250rue Albert Einstein
LEAT BAT 4
LUCIOLES 1
06560 VALBONNE TEL : 04.92.94.28.27

8.2 Rapports de Doctorat

Vous trouverez dans les pages qui suivent les rapports :
- du Pr. Yves Deville
- de M. Amedeo Napoli
- du Pr. Michel Rueher

Ainsi que le rapport de la soutenance.

Développement d'outils algorithmiques pour l'intelligence artificielle Application à la chimie organique

Jean-Charles Régin
Thèse de doctorat
Université Montpellier II

Rapport de thèse

Y. DEVILLE
Université Catholique de Louvain, Belgique
Novembre 1995

Sujet

Le thème initial de cette thèse était les liaisons stratégiques en synthèse organique (chimie organique). Le développement de méthodes d'apprentissage dans ce domaine à rapidement conduit l'auteur à la constatation que la résolution du problème de l'isomorphisme de sous-graphes était un des problèmes majeurs à résoudre. Une grande partie de la thèse (une moitié) est donc consacrée à ce problème et au développement de solutions originales à ce problème important et complexe.

Le sujet de cette thèse est donc double : la résolution du problème de l'isomorphisme de sous-graphes à l'aide de techniques de consistance et la détermination des liaisons stratégiques en synthèse organique.

Mes compétences en chimie organique étant plutôt limitées, ce rapport ne pourra donc donner une évaluation scientifique approfondie de cette seconde partie de la thèse.

Résumé et analyse

Dans la première partie relative à l'isomorphisme des graphes, le problème est d'abord présenté très clairement (Chapitre 2). La NP-complétude du problème est rappelée. Le Chapitre 3 présente brièvement les différentes techniques de résolution existantes pour ce problème. Après analyse de ces méthodes, l'auteur conclut que les méthodes basées sur la propagation de contraintes semblent plus prometteuses. Le reste de cette première partie est donc consacré à cette approche.

Les réseaux de contraintes sont abordés au Chapitre 4. On y découvre comment le problème de l'isomorphisme de graphes peut être représenté comme un problème de satisfaction de contraintes (CSP). Les principales techniques de recherche de solutions sont présentées : la recherche exhaustive, le filtrage a priori, la recherche avec filtrage "forward checking" et la recherche avec filtrage "lookahead". Parmis les méthodes de filtrage, une des plus performante est bien sûr la consistance d'arc (AC). L'auteur présente les différents algorithmes d'AC et principalement le très performant algorithme AC-6. Cet algorithme, développé par Bessière et Cordier, offre une complexité spaciale optimale. De plus, quoi que d'une complexité théorique temporelle identique à AC-4 (AC-4 est optimal en complexité temporelle), il permet de réduire de manière parfois significative le nombre de tests sur les contraintes. Enfin, les aspects heuristiques sur l'ordre des instantiations lors de la recherche sont abordés (ordonnancement des variables et des valeurs). Ce chapitre

est très bien rédigé et complet. Les différents algorithmes sont présentés de manière claire et unifiée.

Le Chapitre 5 est le chapitre le plus important de cette première partie. L'auteur y décrit deux nouveaux algorithmes pour l'arc consistance : AC-7 et AC-Inférence. AC-7 se base sur AC-6, mais améliore les performances en temps d'AC-6. L'idée de base consiste à tirer parti du caractère bidirectionnel des contraintes afin d'éviter de recalculer (inutilement) certaines contraintes déjà calculées précédemment. L'algorithme est expliqué en détail et des preuves d'exactitude et de complexité sont fournies. Une implémentation des structures de données y est également décrite. L'auteur démontre que le gain réalisé par rapport à AC-6 peut être substantiel. Le second nouvel algorithme développé par l'auteur (AC-Inférence), exploite les propriétés des contraintes (reflexivité, commutativité, ...). Sa complexité spaciale est moins bonne que AC-7, mais il peut améliorer en temps AC-7 d'un facteur 2.

Dans ce Chapitre 5, deux nouvelles heuristiques permettant une amélioration de n'importe quel algorithme d'arc consistance sont proposées. La première fusionne les phases d'initialisation et de propagation des algorithmes AC. La seconde suggère de propager les variables dont le domaine est un singleton d'une manière spécifique. Il faut souligner l'important travail d'expérimentation réalisé par l'auteur. Cette expérimentation permet une analyse fine des différences de comportement des différents algorithmes sur diverses classes de problèmes.

Le Chapitre 6 complémente le chapitre précédant en intégrant un algorithme de filtrage (tel que AC) à la procédure de recherche d'une solution. Il s'agit ici de maintenir une propriété de consistance tout au long de la recherche (algorithme MAC). Selon le choix de l'agorithme de consistance, différentes versions sont ainsi proposées; notamment MAC-6, MAC-7 et MAC-Inférence. Tous ces algorithmes sont originaux. Une expérimentation ainsi qu'une analyse détaillée sont présentes. Cette expérimentation traite entre autre le problème de l'isomorphisme de graphes. L'auteur démontre que ces algorithmes ont des comportements plus intéressants que les versions précédemment publiées (MAC-4, basé sur AC-4).

Dans le Chapitre 7, l'auteur traite les contraintes de différence. Il propose une implémentation spécifique et efficace, notamment dans le problème de l'isomorphisme de graphes. Cette méthode se base sur la consistance d'arc généralisée. Le Chapitre 9 traite les graphes étiquetés.

La seconde partie du texte aborde la déterminaison des liaisons stratégiques des molécules en synthèse organique. Le problème est présenté au Chapitre 2. La résolution de ce problème est ici abordée en se basant sur l'apprentissage conceptuel (présenté au Chapitre 3). L'auteur développe ensuite (Chapitre 4) une nouvelle méthode d'apprentissage CNN, base de son travail. Le Chapitre 5 aborde la modélisation du problème d'apprentissage des liaisons stratégiques. L'auteur a notamment dû adapter un algorithme de recherche de cliques maximales. Enfin le Chapitre 6 présente les résultats obtenus par la nouvelle méthode développée par l'auteur.

Contributions

Je voudrais tout d'abord souligner la démarche scientifique de l'auteur. En effet, tous les développements, tant théoriques, algorithmiques qu'expérimentaux, ont une finalité claire : la construction d'outils permettant la résolution du problème de l'isomorphisme de sous-graphes. Ce dernier problème a lui-même une finalité car il est essentiel dans le contexte de l'apprentissage en synthèse des molécules organiques.

Une première contribution de cette thèse est relative à la construction de nouveaux algorithmes de

consistance d'arc : AC-7 et AC-Inférence. Cette contribution est importante car elle améliore les techniques existantes de maintien de la consistance d'arc. Les heuristiques proposées afin d'améliorer tout algorithme d'arc-consistance sont également originales et trés intéressantes car générales et efficaces. Ces contributions sont à mon avis substantielles.

L'intégration de ces nouveaux algorithmes d'arc-consistance dans une procédure de recherche forme une seconde contribution de ce travail. L'auteur prouve que le maintient de la consistance d'arc constitue maintenant une approche efficace. On peut regretter que le texte que nous avons lu n'aborde que très brièvement les techniques dites de "backtracking intelligent" (backjumping), surtout que des résultats originaux ont été obtenus par l'auteur dans ce domaine.

Le traitement spécifique des contraintes de différence ainsi que le traitement de graphes étiquetés montre que l'auteur a su développer de nouvelles techniques specifiques permettant une amélioration substantielle à la résolution du problème de l'isomorphisme de graphes.

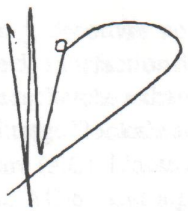
Dans la seconde partie de cette thèse, l'approche et la démarche de l'auteur sont originales. La méthode d'apprentissage proposée est nouvelle. Bien que mes compétances en chimie organique soient limitées, il semble clair que l'auteur a une parfaite maîtrise du sujet. Il réussi à exploiter et à étendre des techniques d'intelligence artificielle et de théorie des graphes pour mieux résoudre les problèmes posés. On pourrait peut-être regretter le peu de de connections explicites, au sein du texte, entre les deux parties de cette thèse.

Finalement, je voudrais remarquer le très large spectre des thèmes abordés par l'auteur. Dans chaque domaine, celui-ci inove et propose des solutions originales, contribuant ainsi significativement à ces domaines.

Conclusion

En conclusion, ce travail est excellent. Chacune des deux parties représente, à elle seule, quasiment une thèse complète. Il est remarquable de constater la maîtrise de l'auteur dans des domaines aussi divers que les techniques de consistance, la théorie des graphes, la chimie organique ou les techniques d'apprentissage. Cette thèse contribue de manière significative à ces différents domaines.

Ce travail est un très bel exemple de développement et d'utilisation de techniques d'intelligence artificielle pour la résolution de problèmes complexes.



Yves Deville 29.11.95

Amedeo Napoli
Chargé de recherches CNRS
CRIN CNRS – INRIA Lorraine
B.P. 239
54506 Vandœuvre-lès-Nancy Cedex

Rapport sur le mémoire intitulé *Développement d'outils algorithmiques pour l'intelligence artificielle. Application à la chimie organique.* présenté par monsieur Jean-Charles Régin pour obtenir le titre de Docteur de l'Université des Sciences et Techniques du Languedoc (Montpellier II), dans la spécialité informatique.

Le travail de thèse de Jean-Charles Régin a pour sujet l'étude et la mise en œuvre d'outils algorithmiques pour l'intelligence artificielle. Ces outils traitent du problème fondamental qu'est l'appariement de structures, qui se ramène plus précisément au problème de l'isomorphisme de sous-graphes ; ils sont appliqués à des graphes moléculaires dans le cadre de la synthèse en chimie organique. Ainsi, le mémoire de thèse comporte deux parties biens distinctes. La première partie traite du problème général de l'isomorphisme de sous-graphes. La seconde partie a pour objet l'étude de méthodes de généralisation en apprentissage symbolique, et l'application de ces méthodes aux graphes moléculaires.

la première partie est riche de huit chapitres, et c'est de loin la plus importante. Le premier chapitre présente le problème de l'isomorphisme de sous-graphes dans sa généralité – isomorphisme de sous-graphes et de sous-graphes partiels, et c'est essentiellement le second qui est considéré dans la suite – tout en évoquant la complexité de ce problème, qui est NP-complet. Vient ensuite la présentation de techniques de résolution du problème de l'isomorphisme de sous-graphes, techniques générales, mais aussi techniques spécifiques employées dans les applications portant sur les graphes moléculaires : isomorphisme de sous-arbres (qui est de complexité polynomiale), recherche de couplages, recherche de cliques maximales, et transformation en réseau de contraintes à satisfaire.

le chapitre 4 décrit les réseaux de contraintes, les méthodes de résolution (ou de satisfaction) associées, et leur application au problème de l'isomorphisme de sous-graphes. Les algorithmes de satisfaction de réseaux de contraintes connaissent à l'heure actuelle une grande popularité, ce qui est justifié, car ils proposent une vision alternative de problèmes difficiles comme celui de l'isomorphisme de sous-graphes, et ils offrent des solutions efficaces et élégantes. La satisfaction d'un réseau de contraintes s'appuie sur deux étapes principales : une étape d'instanciation des variables du réseau (constitution de l'espace dit de recherche du réseau), et une étape de filtrage, qui permet de vérifier la validité d'une instanciation. La consistance d'arc, qui est une technique classique de filtrage, est introduite et discutée sous ses diverses formes : en particulier sont décrits l'algorithme classique AC-4 et l'algorithme plus récent AC-6. Dans le chapitre 5, la consistance d'arc est présentée à la lumière des améliorations apportées par l'auteur : sont proposés notamment deux nouveaux algorithmes, AC-7 et AC-INFÉRENCE, qui ont de bien meilleures performances que leurs prédécesseurs, AC-4 ou même AC-6. Les chapitres 6 et 7, qui sont, comme le chapitre 4, assez techniques, présentent toute une série d'améliorations pour les algorithmes de satisfaction de réseaux de contraintes, qui concernent donc AC-6, AC-7 et AC-INFÉRENCE, et qui permettent d'augmenter encore leur efficacité : la prise en compte de la consistance d'arc pendant la phase d'instanciation (algorithmes MAC-7 et MAC-INFÉRENCE) ; la prise en compte des contraintes

de différences, qui stipulent que les valeurs des variables dans une instanciation sont toutes différentes (algorithme ACRDIFF).

Les chapitres 5, 6 et 7 constituent le cœur de la première partie de la thèse. Le huitième et dernier chapitre de la première partie traite de l'adaptation des algorithmes précédents à la manipulation de graphes étiquetés, ce qui est naturel dans le contexte, puisque les graphes moléculaires sont des cas particuliers de graphes étiquetés. Trois annexes ferment la première partie, dont une relate la mise en œuvre des techniques de satisfaction de réseaux de contraintes sur le célèbre problème du zèbre.

Dans la deuxième partie est abordé le problème de la recherche de liaisons stratégiques dans des molécules, dans le cadre de la synthèse en chimie organique. Une liaison stratégique est une liaison créée au cours de la dernière étape d'une synthèse. Cette "dernière" liaison est celle qu'il vaut mieux déconnecter "en premier" lorsque le problème de la synthèse est considéré sous l'angle de la simplification de la molécule à construire, appelé aussi *approche rétrosynthétique* : en termes de résolution de problèmes, la synthèse repose sur le chaînage avant alors que l'approche rétrosynthétique repose sur le chaînage arrière.

Le problème de la synthèse en chimie organique est survolé dans le chapitre 2 de la deuxième partie. Vient ensuite la présentation de deux méthodes d'apprentissage, en l'occurrence une méthode de voisinage (recherche des plus proches voisins d'un exemple), et ANNA, une méthode d'apprentissage à partir d'exemples symbolique-numérique, qui manipule des énoncés symboliques et dont le processus de décision (d'appartenance d'un nouvel exemple à une classe d'exemples) s'appuie sur des critères numériques. Le chapitre 4 contient une présentation de la méthode baptisée CNN, qui tente de produire des règles expliquant une série d'exemples et de contre-exemples donnés, en s'appuyant sur une méthode de voisinages. Là encore, les résultats sont bons et montrent le bien fondé de la méthode CNN. L'application de CNN à la recherche de liaisons stratégiques dans les graphes moléculaires fait l'objet du chapitre 5. Cette application se sert, entre autres, des méthodes de satisfaction de contraintes exposées dans la première partie, pour la mise en œuvre de l'appariement de structures moléculaires. Les résultats obtenus sont décrits dans le dernier chapitre de la deuxième partie de la thèse. Ces résultats sont encourageants, et ils montrent que des recherches plus approfondies méritent d'être conduites dans cette direction.

Deux annexes ferment la deuxième partie de la thèse: la première contient l'ensemble des exemples sur lesquels a été appliquée la méthode CNN ; la seconde contient un article en soumission ayant pour objet le système d'aide à la conception de synthèses en chimie organique nommé RESYN, système à l'élaboration duquel Jean-Charles Régin a participé activement, mais dont il n'est pas question dans le mémoire de thèse.

Sur la forme du mémoire, il peut être reproché certaines choses à l'auteur. Si le style est alerte et le texte facile à lire par endroits, le style peut être lourd et le texte opaque à d'autres endroits, à cause du manque d'explications et de précisions (et quelquefois à cause du manque de ponctuation). Ainsi, certains paragraphes, algorithmes et figures auraient mérité plus amples explications et commentaires. Plus généralement, le lecteur a l'impression que le bon niveau de discours n'a pas été trouvé: ce qui est important est au même niveau que ce qui ne l'est pas, et en ressort une certaine monotonie. Il aurait fallu être plus judicieux sur le partage entre texte et annexes, et bien sûr, élaguer au besoin : la qualité littéraire devrait valoir la qualité scientifique, et 380 pages (!) n'étaient peut être pas indispensables.

Si la forme n'est pas exempte de reproches, le fond, lui, est plutôt exemplaire. La qualité et la quantité de travail présentées par l'auteur sont impressionnantes : il suffit de considérer la conception des algorithmes AC-7, AC-INFÉRENCE et de leurs variantes, leurs mises en

œuvre et leurs performances, pour se rendre compte qu'un saut qualitatif a été effectué dans l'étude de la consistance d'arc et des méthodes de satisfaction de réseaux de contraintes. Plus encore, le problème de l'appariement de structures, au moins dans le cadre des structures moléculaires, se voit traité de façon rigoureuse et complète, et il se voit associé des algorithmes efficaces : la thèse devrait devenir une référence en la matière.

Il ne faut pas oublier la seconde partie du mémoire, où est abordé le problème difficile de la mise en valeur de stratégies de synthèse, par l'intermédiaire de méthodes d'apprentissage. Là encore, la mise au point de la méthode CNN ouvre de nombreuses perspectives pour l'apprentissage à partir d'exemples, et pas seulement dans le domaine de la synthèse en chimie organique.

De mon point de vue, cette thèse possède un caractère scientifique relativement exceptionnel, qui compense assez largement les problèmes soulevés pas la forme du document. J'espère que l'auteur pourra continuer à travailler dans le cadre qui est défini dans cette thèse. En m'appuyant sur les arguments qui précèdent, je suis tout à fait favorable à ce que les travaux présentés dans ce mémoire par monsieur Jean-Charles Régin lui permettent d'obtenir le titre de Docteur de l'Université des Sciences et Techniques du Languedoc (Montpellier II), dans la spécialité informatique.

Nancy, le 29 novembre 1995
Amedeo Napoli

LABORATOIRE
i3S
INFORMATIQUE
SIGNAUX ET SYSTEMES
DE SOPHIA ANTIPOLIS

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Rapport sur le mémoire de thèse de Monsieur Jean-Charles REGIN, intitulé :

"Développement d'outils algorithmiques pour l'intelligence artificielle. Application à la chimie organique"

L'objectif de la thèse de Monsieur Jean-Charles Regin était d'expliquer le caractère stratégique des liaisons en synthèse organique. Ce problème se formalise par la recherche d'un isomorphisme entre un graphe et un sous graphe (ou un sous graphe partiel), problème difficile que Monsieur Jean-Charles Regin a cherché à résoudre à l'aide des techniques de satisfaction de contraintes. Il a ainsi été amené à proposer différents algorithmes de filtrage et de recherche qui constituent une ***contribution de tout premier plan*** pour la résolution de problèmes de satisfaction de contraintes (CSP). Ces résultats sont présentés dans la première partie de son manuscrit.

Le chapitre I situe clairement le contexte de travail et les principales contributions de l'auteur. Dans le deuxième chapitre Monsieur Jean-Charles Regin rappelle la complexité du problème de l'isomorphisme de graphes partiels et montre bien où se situe la frontière entre les instances polynomiales et NP-complètes de ce problème. Le chapitre suivant présente différentes techniques de résolution pour ces problèmes. L'auteur met en évidence la similitude entre les filtrages effectués par l'algorithme de Ullmann et celui de Sussenguth pour des graphes non orientés et non étiquetés.

Le chapitre 4 aborde l'étude des réseaux de contraintes. L'auteur rappelle brièvement quelques définitions concernant les CSP avant de montrer comment les problèmes de l'isomorphisme à un sous graphe et de l'isomorphisme à un graphe partiel peuvent se modéliser par un réseau de contraintes. Monsieur Jean-Charles Regin effectue ensuite une étude approfondie des principaux algorithmes de recherche et de filtrage utilisés dans les CSP. La pertinence de l'analyse des algorithmes de filtrage AC4 et AC6, ainsi que celle des stratégies de recherche Forward Checking et Look-ahead, témoigne d'une très grande maîtrise de l'ensemble des problèmes liés à la résolution des systèmes de contraintes sur les domaines finis.
Monsieur Jean-Charles Regin montre ensuite comment les caractéristiques spécifiques des problèmes d'isomorphisme à un sous graphe (ou à un graphe partiel) peuvent être exploitées pour améliorer les performances de ces algorithmes. L'auteur explicite avec rigueur le processus d'expérimentation qui sera utilisé pour évaluer les différents algorithmes présentés dans sa thèse.

Dans le chapitre 5, Monsieur Jean-Charles Regin présente deux nouveaux algorithmes de filtrage par consistance d'arc, AC7 et AC-Inference, ainsi que deux heuristiques pour optimiser ce type d'algorithme.

L'algorithme AC7 —developpé en collaboration avec Monsieur Christian Bessière— cherche à exploiter la bi-directionnalité des contraintes pour minimiser le nombre de tests de consistance. Le

LABORATOIRE
i3S
INFORMATIQUE
SIGNAUX ET SYSTEMES
DE SOPHIA ANTIPOLIS

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

principe de base consiste à rechercher un support pour une valeur a sur une contrainte C dans la liste des valeurs supportées par a pour C. Bien que les complexités en temps et en espace ne soient pas meilleures que celle de AC6 dans le pire des cas, AC7 peut permettre d'éviter jusqu'a $O(d^2)$ tests de consistance pour des domaines de taille d.
AC-inférence généralise l'idée de AC7 en cherchant à utiliser d'autres propriétés des contraintes (i.e., commutativité et reflexivité) pour n'effectuer que certains tests de consistance et en déduire automatiquement les autres résultats. La complexité en espace de AC6 n'est toutefois plus conservée.
Des structures de données astucieuses, basées sur des listes chainées et des mécanismes d'indirection, sont proposées pour ces deux algorithmes.

Monsieur Jean-Charles Regin présente ensuite une heuristique pour une meilleure articulation des phases d'initialisation et de propagation des algorithmes de filtrage par consistance d'arc. L'idée de base est d'étudier immédiatement les conséquences de la suppression d'une valeur pour éviter d'effectuer des tests de consistance avec des valeurs non viables.
Une heuristique pour améliorer la phase de propagation lorsque le domaine d'une variable se réduit à un singleton est également proposée par l'auteur. Le traitement effectué fournit un support pour la prise en compte des contraintes de différence lors d'un filtrage par consistance d'arc.

Les résultats expérimentaux obtenus sur des problèmes aléatoires ainsi que sur différentes instances du problème d'allocation de fréquences radio sont encourageants et confirment l'intérêt de ces algorithmes et heuristiques. Aucun gain significatif n'a toutefois été obtenu pour les problèmes d'isomorphisme de sous graphes partiels.

Dans le chapitre 6, Monsieur Jean-Charles Regin remet en cause le principe d'une utilisation d'un filtrage minimal durant la procédure de recherche. Il propose une démarche basée sur le maintien d'un filtrage par consistance d'arc avant toute augmentation (i.e., instanciation d'une nouvelle variable) et après toute réfutation. Ceci requiert la modification de la phase de propagation des algorithmes de filtrage par consistance d'arc pour permettre une mémorisation des informations nécessaires au rétablissement d'un état du réseau équivalent à l'état où celui-ci se trouvait avant une augmentation. Les modifications nécessaires pour les algorithmes AC6, AC7 et AC-inférence sont détaillées. Ces nouveaux algorithmes permettent des gains de performance très significatifs pour l'ensemble des problèmes étudiés (y compris les problèmes d'isomorphisme de sous graphes partiels). Ces résultats expérimentaux valident indéniablement l'approche préconisée par l'auteur.

Le chapitre 7 est consacré à l'étude des contraintes de différence, Monsieur Jean-Charles Regin présente une implémentation très efficace pour les contraintes de différence de la consistance d'arc généralisée. Le filtrage d'une contrainte de différence est basée sur la recherche d'un couplage maximal dans le graphe des valeurs associé à cette contrainte. Le mécanisme de propagation repose sur la suppression des arêtes n'appartenant à aucun couplage couvrant. L'auteur introduit deux nouvelles consistances : la "consistance d'arc généralisée pour les contraintes de différence" et la "consistance d'arc pour les contraintes binaires respectant les contraintes de différence". L'algorithme $AC_{\neq}$ combine ces deux consistances. Les performances de $AC_{\neq}$ pour des problèmes réels d'isomorphisme de sous graphes partiels sont spectaculaires. $AC_{\neq}$ est de mon point de vue un des résultats les plus

LABORATOIRE D'INFORMATIQUE, SIGNAUX ET SYSTEMES DE SOPHIA ANTIPOLIS

I3S-U.R.A.1376-C.N.R.S.-UNIVERSITE DE NICE-SOPHIA ANTIPOLIS 650,Route des Colles,B.P. 145 - 06903 SOPHIA ANTIPOLIS Cedex, FRANCE
Tél: (33) 92 96 51 50 Fax: (33) 92 96 51 55

LABORATOIRE
I3S
INFORMATIQUE
SIGNAUX ET SYSTEMES
DE SOPHIA ANTIPOLIS

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS
CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

prometteurs de Monsieur Jean-Charles Regin.

Dans la deuxième partie du manuscrit Monsieur Jean-Charles Regin présente un système d'apprentissage basé sur la recherche de ressemblances entre molécules. L'auteur explique d'abord l'intérêt de la reconnaissance de liaisons stratégiques en chimie organique avant de présenter les principes de base des "méthodes d'apprentissage de concepts à partir d'exemples". La méthode CNN (*conceptual nearest neighbour*) qu'il propose est d'abord expérimentée sur des problèmes généraux. Monsieur Jean-Charles Regin effectue ensuite une modélisation de l'apprentissage des liasons en chimie organique qui autorise la prise en compte de particularités du domaine d'application (e.g., les effets électroniques). Le mécanisme de généralisation utilisé fait appel à un algorithme de recherche de cliques maximales.
Le système prototype qui a été développé obtient d'excellents résultats pour les exemples d'apprentissage. L'auteur reconnait toutefois que des expérimentations plus complètes sont nécessaires pour réellement valider l'approche préconisée.

Avis du rapporteur:

Les résultats obtenus par Monsieur Jean-Charles Regin dans le domaine de la programmation par contraintes sont ***originaux et importants***. Je suis moins à même de juger sa contribution dans la domaine de l'apprentissage mais l'approche utilisée dans son "système d'apprentissage conceptuel par données symboliques" me parait prometteuse. Cet ensemble de résultats est d'autant plus impressionnant que les problèmes traités sont difficiles et que les contributions sont validées expérimentalement sur de véritables applications.

Le manuscrit est très bien rédigé et montre que Monsieur Jean-Charles Regin possède une grande culture informatique. J'ai particulièrement apprécié la présentation systématique des "intuitions" à la base des algorithmes avant leur description formelle.

Le travail réalisé est plus que conséquent et ***je recommande très vivement*** que Monsieur Jean-Charles Regin soit autorisé à présenter son mémoire en vue d'obtenir le titre de Docteur en Informatique.

Le 28 Novembre 1995

Michel Rueher

Professeur à l'Université de Nice - Sophia Antipolis

LABORATOIRE D'INFORMATIQUE, SIGNAUX ET SYSTEMES DE SOPHIA ANTIPOLIS

I3S-U.R.A.1376-C.N.R.S.-UNIVERSITE DE NICE-SOPHIA ANTIPOLIS 650,Route des Colles,B.P. 145 - 06903 SOPHIA ANTIPOLIS Cedex, FRANCE
Tél: (33) 92 96 51 50 Fax: (33) 92 96 51 55

— Diplôme de DOCTORAT —

de la thèse de

M. Jean Charles Regin

Date : 21 décembre heure : 10h30

J.C. Régin a magistralement exposé son travail de recherche. Il a réalisé une synthèse, très vivante, procédant à l'envers du manuscrit, i.e. partant du problème initial de chimie organique (extraction automatique de règles de synthèse de molécules) pour aboutir aux algorithmes de résolution.

Cette sélection de ses travaux nous a permis d'apprécier le niveau d'excellence atteint.

Constatant l'ampleur du domaine abordé, de la reconnaissance internationale des travaux, du transfert déjà réalisé (une partie des résultats est déjà utilisée dans un cadre industriel), le jury unanime tient à souligner le caractère exceptionnel de cette thèse.

TSVP →

Le jury, après avoir délibéré, a décerné à M. Jean Charles Régin le grade de Docteur de l'Université de Montpellier II, avec la mention Très honorable avec les félicitations du jury

Montpellier, le 21 décembre 95

— LE JURY —

NOM et Prénom	signature
BESSIERE Christian	
PUGET Jean-François	
LAURENÇOT Claude	
Amedeo NAPOLI	
CASTRO Bertrand	
GASQUEL Olivier	
RUEHER Michel	
OU INQUETON Joël	

LE PRESIDENT du JURY

M. Haï

Devant la qualité exceptionnelle de cette thèse, malgré la décision de la commission des thèses en informatique de ne plus attribuer de mention « T.H. avec Félicitations du jury », décision pas encore validée par l'École Doctorale SPI et le conseil scientifique, le jury attribue à J.C. Regin "les félicitations du jury".

LABORATOIRE D'INFORMATIQUE DE ROBOTIQUE
ET DE MICROELECTRONIQUE DE MONTPELLIER
UMR CNRS / Université Montpellier II n° 9928

LIRM MONTPELLIER

(1)

le 22 décembre 95

Rapport sur la mention : "Félicitations du Jury"
pour la thèse de J.C. Régin

Voici brièvement quelques uns des arguments qui justifient la décision du jury.

(1) Tout d'abord, la méthode de travail, exemplaire. Partant d'un problème d'apprentissage automatique en chimie organique et devant les faibles performances de son premier prototype, il a utilisé une modélisation basée sur les systèmes à contraintes. Ce faisant il est devenu l'un des meilleurs spécialistes mondiaux du domaine, cosignant un article avec Freuder. Cette dernière modélisation lui a permis de résoudre son problème initial efficacement.

UNIVERSITE MONTPELLIER II

161 rue Ada - 34392 Montpellier Cedex 5 - FRANCE
Tél : (33) 67 41 85 85 - Fax : (33) 67 41 85 00

CNRS

(2)

(2) Ses algorithmes publiés ont déjà été repris et intégrés dans les logiciels de la Société Ilog (leader mondial en matière de systèmes à contraintes).

(3) Ses travaux ouvrent de nombreuses perspectives et seront utilisés dans le cadre de nos applications en informatique appliquée à la chimie (GDR CNRS TICCO).

(4) Son manuscrit est bien écrit et très complet.

Enfin pour conclure, J. C. Regin, sera embauché par la société Ilog, dès janvier 96, confirmant ainsi l'intérêt pratique des théories qu'il a développé au cours de sa thèse.

M Haeh

Président du jury

Chapitre 9

Liste de Publications jointes

Vous trouverez dans les pages suivantes une sélection de publications :

• Le document de Synthèse des Recherches de mon HDR. Afin d'économiser de la place ce document est imprimé sur deux pages. Vous pouvez facilement accéder au document original qui se trouve à l'adresse web :
www.constraint-programming.com/people/regin/papers/hdrsynthese.pdf

• L'article "A filtering algorithm for constraints of difference in CSPs", J-C. Régin, **AAAI-94**, Seattle, WA, USA, pp 362–367, 1994. Cet article est accessible par internet à l'adresse :
www.constraint-programming.com/people/regin/papers/alldiff.pdf

• L'article "An Optimal Coarse-grained Arc Consistency Algorithm", C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang, **Artificial Intelligence**, vol 165 (2), pp 165–185, 2005. Cet article est accessible par internet à l'adresse :
www.constraint-programming.com/people/regin/papers/aij05-ac2001.pdf

• L'article "Inequality-sum : a global constraint capturing the objective function", J-C. Régin, M. Rueher, **RAIRO Operations Research**, 39, pp 123-139, 2005. Cet article est accessible par internet à l'adresse :
www.constraint-programming.com/people/regin/papers/ineqsum05.pdf

• La table des matières du livre "Modeling Patterns in Constraint Programming", que je suis en train d'écrire

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 produit ; 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.

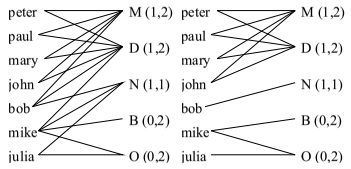


FIG. 2.2 – Un exemple de contrainte globale de cardinalité.

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 derniers 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 $\forall x \in XD(x) = \{a, b\}$;
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* $X(C)$
$\quad$ *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 $\sum_{i=1}^{|X(C)|} cost(var(C, i), \tau[i]) \leq H \}$
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 $f^o_{ab} > 0$. 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 $T(IS)$:*

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

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*

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

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 $(x = nx \text{ xor } nx = e(S_i))$ 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 :*

$T(C) = \{ \quad \tau$ *où* τ *est un tuple de* $X(C)$
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, vérifient $v(Q) \geq 1$.

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

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 calcules.

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 $Capa^k_{ij}, 1 \leq k \leq K_{ij}$, sont données. Nous posons par ailleurs $Capa^0_{ij} = 0$. 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 $Wmin^k_{ij}$ et une valeur maximale $Wmax^k_{ij}$ données. Le problème de dimensionnement consiste donc non seulement à choisir pour chaque arc (i, j) une capacité $Capa^k_{ij}$, mais aussi à choisir le coefficient multiplicateur w^k_{ij} retenu. Notons que la capacité $Capa^k_{ij}$ doit nécessairement être retenue dès lors que $Wmin^k_{ij}$ est supérieure ou égale à 1. Il ne peut donc exister qu'un seul k avec $Wmin^k_{ij} > 0$, 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 $Capa^k_{ij}$ est retenue avec un coefficient multiplicateur $w^k_{ij} > 0$ pour l'arc (i, j), alors la k-ième capacité $Capa^k_{ji}$ doit être retenue pour l'arc (j, i), avec le même coefficient multiplicateur $w^k_{ji} = w^k_{ij}$.

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 $Cost^k_{ij}$, 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 $Risk^k_{ij}$ 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 $Risk^k_{ij} = 1$.

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.

A filtering algorithm for constraints of difference in CSPs *

Jean-Charles RÉGIN

GDR 1093 CNRS
LIRMM UMR 9928 Université Montpellier II / CNRS
161, rue Ada – 34392 Montpellier Cédex 5 – France
e-mail : regin@lirmm.fr

Résumé

Many real-life Constraint Satisfaction Problems (CSPs) involve some constraints similar to the alldifferent constraints. These constraints are called constraints of difference. They are defined on a subset of variables by a set of tuples for which the values occuring in the same tuple are all different. In this paper, a new filtering algorithm for these constraints is presented. It achieves the generalized arc-consistency condition for these non-binary constraints. It is based on matching theory and its complexity is low. In fact, for a constraint defined on a subset of p variables having domains of cardinality at most d, its space complexity is $O(pd)$ and its time complexity is $O(p^2d^2)$. This filtering algorithm has been successfully used in the system RESYN [Vismara *et al.*, 1992], to solve the subgraph isomorphism problem.

1 Introduction

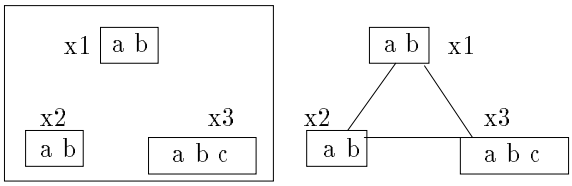
The constraint satisfaction problems (CSPs) form a simple formal frame to represent and solve some problems in artificial intelligence. The problem of the existence of solutions in a CSP is NP-complete. Therefore, some methods have been developed to simplify the CSP before or during the search for solutions. The consistency techniques are the most frequently used. Several algorithms achieving arc-consistency have been proposed for binary CSPs [Mackworth, 1977; Mohr and Henderson, 1986; Bessière and Cordier, 1993; Bessière, 1994] and for n-ary CSPs [Mohr and Masini, 1988a]. Only limited works have been carried out on the semantics of constraints : [Mohr and Masini, 1988b] have described

*This work was supported by SANOFI-CHIMIE

an improvement of the algorithm AC-4 for special constraints introduced by a vision problem, [Van Hentenryck *et al.*, 1992] have studied monotonic and functional binary constraints. In this work, we are interested in a special case of n-ary constraints : the constraints of difference, for which we propose a filtering algorithm.

A constraint is called *constraint of difference* if it is defined on a subset of variables by a set of tuples for which the values occuring in the same tuple are all different. They are present in many real-life problems.

These constraints can be represented as n-ary constraints and filtered by the generalized arc-consistency algorithm GAC4 [Mohr and Masini, 1988a]. This filtering efficiently reduces the domains but its complexity can be expensive. In fact, it depends on the length and the number of all admissible tuples. Let us consider a constraint of difference defined on p variables, which take their values in a set of cardinality d. Thus, the number of admissible tuples corresponds to the number of permutations of p elements selected from d elements without repetition : ${}^dP_p = \frac{d!}{(d-p)!}$. Therefore some constraint resolution systems, like CHIP [Van Hentenryck, 1989], represent these n-ary constraints by sets of binary constraints. In this case, a binary constraint of difference is built for each pair of variables belonging to the same constraint of difference. But the pruning performance of arc-consistency, for these constraints is poor. In fact, for a binary alldifferent constraint between two variables i and j, arc-consistency removes a value from domain of i only when the domain of j is reduced to a single value. Let us suppose we have a CSP with 3 variables x_1, x_2, x_3 and one



Representation by 3-ary constraint

Representation by binary constraints of difference

Figure 1:

constraint of difference between these variables (see figure 1). The domains of variables are $D_1 = \{a, b\}$, $D_2 = \{a, b\}$ and $D_3 = \{a, b, c\}$. The GAC4 filtering with the constraint of difference represented by a 3-ary constraint, removes the values a and b from the domain of x_3, while arc-consistency with the constraint of difference represented by binary constraints of difference, does not delete any value.

In this paper we present an efficient way of implementing the generalized

arc-consistency condition for the constraints of difference, in order to benefit from its pruning performances. Its space complexity is in $O(pd)$ and its time complexity is in $O(p^2d^2)$.

The rest of the paper is organized as follows. Section 2 gives some preliminaries on constraint satisfaction problems and matching, and proposes a restricted definition of arc-consistency, which concerns only the constraints of difference : the diff-arc-consistency. Section 3 presents a new condition to ensure the diff-arc-consistency in CSPs having constraints of difference. In section 4 we propose an efficient implementation to achieve this condition and analyse its complexity. In section 5, we show its performance and its interest with an example. A conclusion is given in section 6.

2 Preliminaries

A finite CSP (Constraint Satisfaction Problem) $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ is defined as a set of n *variables* $X = \{x_1, ..., x_n\}$, a set of finite *domains* $\mathcal{D} = \{D_1, ..., D_n\}$ where D_i is the set of possible *values* for variable i and a set of *constraints* between variables $\mathcal{C} = \{C_1, C_2, ..., C_m\}$. A constraint C_i is defined on a set of variables $(x_{i_1}, ..., x_{i_j})$ by a subset of the cartesian product $D_{i_1} \times ... \times D_{i_j}$. A solution is an assignment of value to all variables which satisfies all the constraints. We will denote by :

- $D(X')$ the union of domains of variables of $X' \subseteq X$ ($i.e\ D(X') = \cup_{i \in X'} D_i$).
- X_C the set of variables on which a constraint C is defined.
- p the arity of a constraint C : $p = |X_C|$.
- d the maximal cardinality of domains.

A value a_i in the domain of a variable x_i is consistent with a given n-ary constraint if there exists values for all the other variables in the constraint such that these values with a_i *together simultaneously* satisfy the constraint. More generally, arc-consistency for n-ary CSPs or the generalized arc-consistency is defined as follows [Mohr and Masini, 1988a]:

Definition 1 *A CSP* $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ *is* **arc-consistent** *iff :* $\forall x_i \in X, \forall a_i \in D_i, \forall C \in \mathcal{C}$ *constraining* $x_i, \forall x_j, ..., x_k \in X_C, \exists a_j, ..., a_k$ *such that* $C(a_j, ..., a_i, ..., a_k)$ *holds.*

Definition 2 *Given a CSP* $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$*, a constraint* C *is called* **constraint of difference** *if it is defined on a subset of variables* $X_C = \{x_{i_1}, ..., x_{i_k}\}$ *by a set of tuples, denoted by* $tuples(C)$ *such that :* $tuples(C) = D_{i_1} \times ... \times D_{i_k} \setminus \{(d_1, ..., d_k) \in D_{i_1} \times ... \times D_{i_k}\ s.t.\ \exists\, u, v \mid d_u = d_v\}$

From the previous definition, we propose a special arc-consistency which concerns only the constraints of difference :

Definition 3 *A CSP $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ is* **diff-arc-consistent** *iff all of its constraints of difference are arc-consistent.*

Definition 4 *Given a constraint of difference C, the bipartite graph $GV(C) = (X_C, D(X_C), E)$ where $(x_i, a) \in E$ iff $a \in D_i$ is called* **value graph** *of C.*

Figure 2 gives an example of a constraint of difference and its value graph.

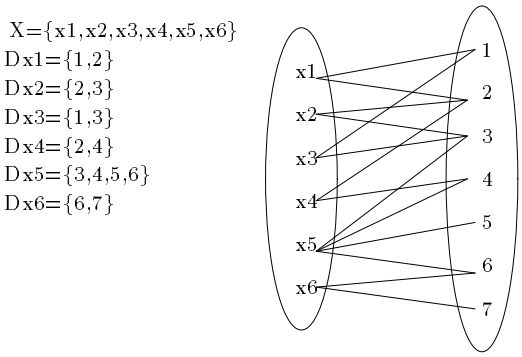


Figure 2: A constraint of difference defined on a set X and its value graph

Definition 5 *A subset of edges in a graph G is called* **matching** *if no two edges have a vertex in common. A matching of maximum cardinality is called a* **maximum matching**. *A* **matching** *M* **covers a set** *X if every vertex in X is an endpoint of an edge in M.*

Note that a matching which covers X in a bipartite graph $G = (X, Y, E)$ is a maximum matching.

From the definition of a matching and the value graph we present, in the next section, a new necessary condition to ensure the diff-arc-consistency in CSPs having constraints of difference.

3 A new condition for CSPs having constraints of difference

The following theorem establishes a link between the diff-arc-consistency and the matching notion in the value graph of the constraints of difference.

Theorem 1 *Given a CSP $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$. $\mathcal{P}$ is diff-arc-consistent iff for each constraint of difference C of $\mathcal{C}$ every edge in $GV(C)$ belongs to a matching which covers X_C in $GV(C)$.*

proof
$\Rightarrow$: Let us consider a constraint of difference C and $GV(C)$ its value graph. From each admissible tuple of C, a set of pairs can be built. A pair consists of a variable and its assigned value in the tuple. The set of pairs contains a pair for each variable. This set corresponds to a set of edges, denoted by A in $GV(C)$. Since $\mathcal{P}$ is diff-arc-consistent, the values in each tuple are all different. Thus, two edges of A cannot have a vertex in common and A is a matching which covers X_C. Moreover, each value of each variable in the constraint belongs to at least one tuple. So, each edge of $GV(C)$ belongs to a matching which covers X_C.
$\Leftarrow$: Let us consider a variable x_i and a value a of its domain. For each constraint of difference C, the pair (x_i, a) belongs to a matching which covers X_C in $GV(C)$. Since in a matching no two edges have a vertex in common, there exists values for all the other variables in the constraint such that these values together simultaneously satisfy the constraint. So $\mathcal{P}$ is diff-arc-consistent. □

The use of matching theory is interesting because [Hopcroft and Karp, 1973] have shown how to compute a matching which covers X in a bipartite graph $G = (X, Y, E)$, with m edges, [1] in time $O(\sqrt{|X|}m)$.

This theorem gives us an efficient way to represent the constraint of difference in a CSP. In fact, a constraint of difference can be represent only by its value graph, with a space complexity in $O(pd)$. It also allows us to define a basic algorithm (algorithm 1) to filter the domains of variables of the set on which one constraint of difference is defined. This algorithm builds the value graph of the constraint of difference and computes a matching which covers X_C in order to delete every edge which belongs to no matching covering X_C. Figure 3 gives an application of this filtering.

Algorithm 1: DIFF-INITIALIZATION(C)
% returns false if there is no solution, otherwise true
% the function COMPUTEMAXIMUMMATCHING(G) computes a maximum matching in the graph G
begin
1 Build $G = (X_C, D(X_C), E)$
2 $M(G) \leftarrow$ COMPUTEMAXIMUMMATCHING(G)
if $|M(G)| < |X_C|$ **then return** *false*
3 REMOVEEDGESFROMG(G,$M(G)$)
return *true*
end

The complexity of step 1 is $O(d|X_C| + |X_C| + |D(X_C)|)$. Step 2 costs $O(d|X_C|\sqrt{|X_C|})$. And we now show that it is possible to compute step 3 in linear time. So the complexity for one constraint of difference will be $O(d|X_C|\sqrt{|X_C|})$.

[1] [Alt *et al.*, 1991] give an implementation of Hopcroft and Karp's algorithm which runs in time $O(|X|^{1.5}\sqrt{m \log |X|})$. For dense graph this is an improvement by a factor of $\sqrt{\log |X|}$.

4 Deletion of every edge which belongs to no matching which covers X

In order to simplify the notation, we consider a bipartite graph $G = (X, Y, E)$ rather than the bipartite graph $G = (X_C, D(X_C), E)$, and a matching M which covers X in G. In order to understand how we can delete every edge which

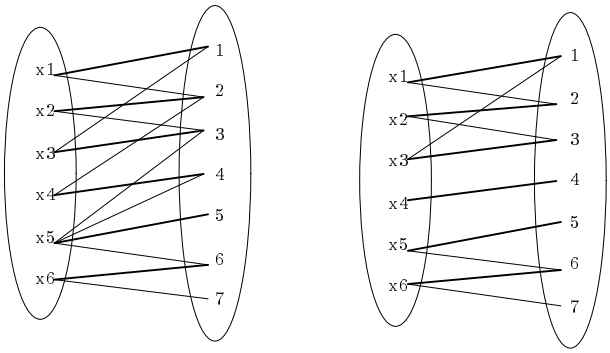


Figure 3: A value graph before and after the filtering.

belongs to no matching, we present a few definitions about matching theory. For more information the reader can consult [Berge, 1970] or [Lovász and Plummer, 1986].

Definition 6 *Let M be a matching, an edge in M is a* **matching** *edge; every edge not in M is* **free**. *A vertex is* **matched** *if it is incident to a matching edge and* **free** *otherwise. An* **alternating path** *or* **cycle** *is a simple path or cycle whose edges are alternately matching and free. The* **length** *of an alternating path or cycle is the number of edges it contains. An edge which belongs to every maximum matching is* **vital**.

Figure 3 gives an example of a matching which covers X in a bipartite graph. The bold edges are the matching edges. Vertex 7 is free. The path $(7, x6, 6, x5, 5)$ is an alternating path which begins at a free vertex. The cycle $(1, x3, 3, x2, 2, x1, 1)$ is an alternating cycle. The edge $(x4, 4)$ is vital.

Property 1 (Berge 1970) *An edge belongs to some of but not all maximum matchings, iff, for an arbitrary maximum matching M, it belongs to either an even alternating path which begins at a free vertex, or an even alternating cycle.*

From this property we can find for an arbitrary matching M which covers X, every edge which belongs to no matching covering X. There are the edges which belong to neither M (there are not vital), nor an even alternating path which begins at a free vertex, nor an even alternating cycle.

Proposition 1 *Given a bipartite graph $G = (X, Y, E)$ with a matching M which covers X and the graph $G_O = (X, Y, Succ)$, obtained from G by orienting edges with the function :*

$\forall x \in X : Succ(x) = \{y \in Y \,/\, (x, y) \in M\}$

$\forall y \in Y : Succ(y) = \{x \in X \,/\, (x, y) \in E - M\}$

we have the two following properties :

1) *Every directed cycle of G_O corresponds to an even alternating cycle of G, and conversely.*

2) *Every directed simple path of G_O, which begins at a free vertex corresponds to an even alternating path of G which begins at a free vertex, and conversely.*

proof

If we ignore the parity, it is obvious that the proposition is true. In the first case, since G is bipartite it does not have any odd cycle. In the second case, we must show every directed simple path of G_O which begins at a free vertex to corresponds to an even alternating path of G which begins at a free vertex. M is a matching which covers X, so there is no free vertex in X. Since G is bipartite and since every path begins at a free vertex, in Y, every odd directed simple path ends with a vertex in X. From this vertex, we can always find a vertex in Y which does not belong to the path, because every vertex in X has one successor and because a vertex in Y has one predecessor. Therefore from an odd directed simple path we can always build an even directed simple path. □

From this proposition we produce a linear algorithm (algorithm 2), that deletes every edge which does not belong to any matching which covers X.

Step 2 finds all edges belonging to the directed simple paths of G_O, which begins at a free vertex. Moreover, it finds some edges belonging to the directed cycles of G_O. Step 3 computes the strongly connected component of G_O, because an edge joining two vertices in the same strongly connected component belongs to a directed cycle and conversely. These edges belong to an even alternating cycle of G (cf point 1 of proposition 1). After this step the set A of all edges belonging to some but not all matchings covering X are known. The set RE of edges to remove from G is: $RE = E - (A \cup M)$. This is done by step 4. The algorithm complexity is the same as the search for strongly connected components[Tarjan, 1972] , *i.e* $O(m+n)$ for a graph with m edges and n vertices.

We have shown how for *one* constraint of difference C every edge which belongs to no matching which covers X_C can be deleted. But a variable can be constrained by several constraints and it is necessary to propagate the deletions. In fact, let us consider x_i a variable of X_C, x_i can be constrained by several constraints. Thus, a value of D_i can be deleted for reasons independent from C. This deletion involves the removal of one edge from $GV(C)$. So, it is necessary to study the consequences of this modification of the $GV(C)$ structure.

Algorithm 2: RemoveEdgesFromG($G, M(G)$)
% RE is the set of edges removed from G.
% $M(G)$ is a matching of G which covers X
% The function returns RE
begin
1 Mark all directed edges in G_O as "unused".
Set RE to $\emptyset$.
2 Perform a breadth-first search starting from
free vertices, and mark all traversed edges as "used".
3 Compute the strongly connected components of G_O.
Mark as "used" any directed edge that joins two
vertices in the same strongly connected component.
4 **for** each directed edge de marked as "unused" **do**
set e to the corresponding edge of de
if $e \in M(G)$ **then** mark e as "vital"
else
$RE \leftarrow RE \cup \{e\}$
remove e from G
return RE
end

5 Propagation of deletions

The deletion of values for one constraint of difference can involve some modifications for the other constraints. And for the other constraints of difference we can do better than repeat the first algorithm by using the fact that before the deletion, a matching which covers X is known.

The propagation algorithm we propose has two sets as parameters. The first one represents the set of edges to remove from the bipartite graph, and the second the set of edges that will be deleted by the filtering. The algorithm needs a function, denoted by MatchingCoveringX(G, M_1, M_2), which computes a matching M_2, which covers X, from a matching M_1 which is not maximum. It returns true if M_2 exists and false otherwise. The new filtering is represented by algorithm 3.

It is divided into three parts. First, it removes edges from the bipartite graph. Second, it eventually computes a new matching which covers X_C. Third, it deletes the edges which does not belongs to any matching covering X_C. The algorithm returns false if ER contains a vital edge or if there does not exist a matching which covers X_C.

Now, let us compute its complexity. Let m be the number of edges of G, and n be the number of vertices. Let us suppose that we must remove k edges from G ($|ER| = k$). The complexity of 1 is in $O(k)$. Step 2 involves, in the worst case, the computation of a matching covering X_C from a matching of cardinality $|M - k|$. This computation has cost $O(\sqrt{k}\, m)$ (see theorem 3 of [Hopcroft and Karp, 1973]). The complexity of step 3 is in $O(m)$.

Algorithm 3: DIFF-PROPAGATION(G,$M(G)$,ER,RE)
% the function returns false if there is no solution
% G is a value graph
% $M(G)$ is a matching which covers X_C
% ER is the set of edges to remove from G
% RE is the set of edges that will be deleted by the filtering
begin
  $computeMatching \leftarrow false$
1 **for** each $e \in ER$ **do**
    if $e \in M(G)$ **then**
      $M(G) \leftarrow M(G) - \{e\}$
      if e is marked as "vital" **then return** $false$
      else $computeMatching \leftarrow true$
    remove e from G
2 **if** $computeMatching$ **then**
    if $\neg$ MATCHINGCOVERINGX(G,$M(G)$,M') **then**
      return $false$
    else
      $M(G) \leftarrow M'$
3 $RE \leftarrow$ REMOVEEDGESFROMG(G,$M(G)$)
  return $true$
end

In the worst case, the edges of G can be deleted one by one. Then the previous function will be called m times. So the global complexity is in $O(m^2)$. If $p = |X_C|$ and d is the maximum cardinality of domains of variables of X_C, then the complexity is in $O(p^2 d^2)$ for one constraint of difference.

6 An example : the zebra problem

1. There are five houses, each of a different color and inhabited by men of different nationalities, with differents pets, drinks and cigarettes.
2. The Englishman lives in the red house.
3. The Spaniard owns a dog.
4. Coffee is drunk in the green house.
5. The Ukrainian drinks tea.
6. The green house is immediately to the right of the ivory house.
7. The Old-Gold smoker owns snails.
8. Kools are being smoked in the yellow house.
9. Milk is drunk in the middle house.
10. The Norwegian lives in the first house on the left.
11. The Chesterfield smoker lives next to the fox owner.
12. Kools are smoked in the house next to the house where the horse is kept.

13. The Lucky-Strike smoker drinks orange juice.
14. The Japanese smokes Parliament.
15. The Norwegian lives next to the blue house.

The *query* is : Who drinks water and who owns the zebra ?

This problem can be represented as a constraint network involving 25 variables, one for each of the five colors, drinks, nationalities, cigarettes and pets :

C_1 red	B_1 coffee	N_1 Englishman	T_1 Old-Gold	A_1 dog
C_2 green	B_2 tea	N_2 Spaniard	T_2 Chesterfield	A_2 snails
C_3 ivory	B_3 milk	N_3 Ukranian	T_3 Kools	A_3 fox
C_4 yellow	B_4 orange	N_4 Norwegian	T_4 Lucky-Strike	A_4 horse
C_5 blue	B_5 water	N_5 Japanese	T_5 Parliament	A_5 zebra

Each of the variables has domain values $\{1,2,3,4,5\}$, each number corresponding to a house position (e.g. assigning the value 2 to the variable *horse* means that the horse owner lives in the second house) [Dechter, 1990]. The assertions 2 to 15 are translated into unary and binary constraints. In addition, there are three ways of representing the first assertion which means that the variables in the same cluster must take different values :

1. A binary constraint is built between any pair of variables of the same cluster ensuring that they are not assigned the same value. In this case we have a binary CSP.

2. Five 5-ary constraints of difference are built (one for each of the clusters). And the CSP is not binary.

3. The five 5-ary constraints of difference are represented by their value graphs. The space complexity of one constraint is in $O(pd)$.

The first representation is generally used to solve the problem [Dechter, 1990; Bessière and Cordier, 1993]. From these three representations we can study the different results obtained from arc-consistency. They are given in figures 4 and 5. The constraints corresponding to the assertions 2 to 15 are represented in extension. The constraints of difference among the variables of each cluster are omitted for clarity.

For the first representation, the result of the filtering by arc-consistency is given in figure 4.

For the second representation, the filtering algorithm employed is the generalized arc-consistency. Figure 5 shows the new results. It has pruned more values that the previous one.

For the third representation, the filtering algorithm employed is arc-consistency for the binary constraints combined with the new filtering for the constraints of difference. The obtained results are the same as with the second method.

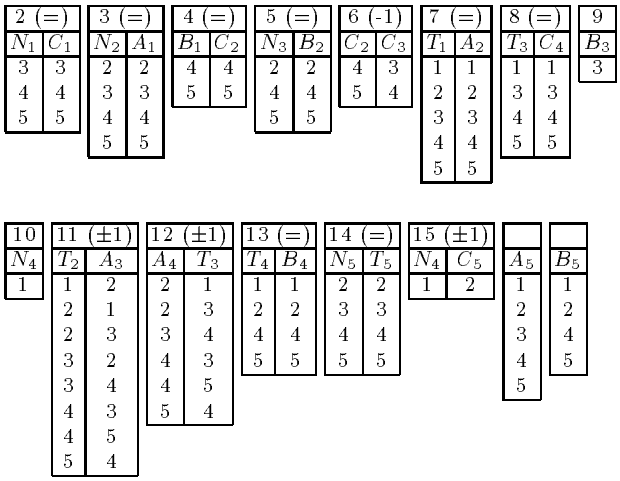


Figure 4:

Let us denote by a the number of binary constraints corresponding to the assertions 2 to 15, p the size of a cluster, c the number of clusters, d the number of values in a domain and $O(ed^2)$ the complexity for arc-consistency[2] in binary CSPs. Let us compute the complexity for the three methods :

1. For the first representation, the number of binary constraints of difference added is in $O(cp^2)$. So, the filtering complexity is $O((a+cp^2)d^2)$.

2. In the second case, we can consider that the complexity is the sum of the lengths of all admissible tuples for the five 5-ary constraints. It is in $O(\frac{d!}{(d-p)!}p)$.

3. For the third method arc-consistency is in $O(ad^2)$ and the filtering for the constraints of difference is in $O(cp^2d^2)$. The total complexity is in $O(ad^2)+O(cp^2d^2)$. It is equivalent to the first one.

The second filtering eliminates more values than the first one. But its complexity is higher. The representation and the algorithm proposed in this paper give pruning results equivalent to the second approach with the same complexity as the first one. So we can conclude that the new filtering is good for problems looking like the zebra problem.

[2][Mohr and Masini, 1988b] reduce this complexity to $O(ed)$ for the binary alldifferent constraints

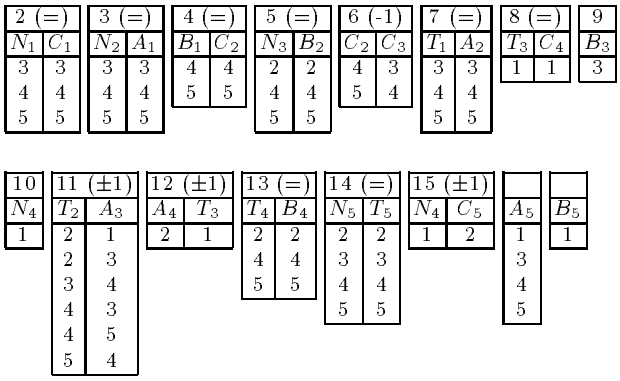


Figure 5:

7 Conclusion

In this paper we have presented a filtering algorithm for constraints of difference in CSPs. This algorithm can be viewed as an efficient way of implementing the generalized arc-consistency condition for a special type of constraint : the constraints of difference. It allows us to benefit from the pruning performance of the previous condition with a low complexity. In fact, its space complexity is in $O(pd)$ and its time complexity is in $O(p^2d^2)$ for one constraint defined on a subset of p variables having domains of cardinality at most d. It has been shown to be very efficient for the zebra problem. And it has been successfully used to solve the subgraph isomorphism problem in the system RESYN [Vismara *et al.*, 1992], a computer-aided design of complex organic synthesis plan.

8 Acknowledgments

We would like to thank particularly Christian Bessière and also Marie-Catherine Vilarem, Tibor Kökény for their comments which helped improve this paper.

Bibliographie

[Alt *et al.*, 1991] H. Alt, N. Blum, K. Melhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $o(n^{1,5}\sqrt{m/\log n})$. *Information Processing Letters*, 37:237–240, 1991.

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

[Bessière and Cordier, 1993] C. Bessière and M.O. Cordier. Arc-consistency and arc-consistency again. In *Proceedings AAAI*, pages 108–113, Washington, DC, 1993.

[Bessière, 1994] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65(1):179–190, 1994.

[Dechter, 1990] R. Dechter. Enhencement schemes for constraint processing : Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

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

[Lovász and Plummer, 1986] L. Lovász and M.D. Plummer. *Matching Theory.* North Holland mathematics studies 121, 1986.

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

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

[Mohr and Masini, 1988a] R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings ECAI*, pages 651–656, 1988.

[Mohr and Masini, 1988b] R. Mohr and G. Masini. Running efficiently arc consistency. *Syntactic and Structural Pattern Recognition*, F45:217–231, 1988.

[Tarjan, 1972] R.E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal of Computing*, 1:146–160, 1972.

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

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

[Vismara *et al.*, 1992] P. Vismara, J-C. Régin, J. Quinqueton, M. Py, C. Laurenço, and L. Lapied. RESYN : Un système d'aide à la conception de plans de synthèse en chimie organique. In *Proceedings 12th International Conference Avignon'92*, volume 1, pages 305–318, Avignon, 1992. EC2.

An Optimal Coarse-grained Arc Consistency Algorithm [*] [†]

Christian Bessiere
LIRMM-CNRS (UMR 5506)
161 rue Ada,
34392 Montpellier Cedex 5, France

Jean-Charles Régin
ILOG
1681 route des Dolines,
06560 Valbonne, France

Roland H.C. Yap
School of Computing,
Natl. Univ. of Singapore,
3 Science Dr. 2, Singapore

Yuanlin Zhang
Department of
Computer Science,
Texas Tech University, USA

Abstract

The use of constraint propagation is the main feature of any constraint solver. It is thus of prime importance to manage the propagation in an efficient and effective fashion. There are two classes of propagation algorithms for general constraints: fine-grained algorithms where the removal of a value for a variable will be propagated to the corresponding values for other variables, and coarse-grained algorithms where the removal of a value will be propagated to the related variables. One big advantage of coarse-grained algorithms, like AC-3, over fine-grained algorithms, like AC-4, is the ease of integration when implementing an algorithm in a constraint solver. However, fine-grained algorithms usually have optimal worst case time complexity while coarse-grained algorithms don't. For example, AC-3 is an algorithm with non-optimal worst case complexity although it is simple, efficient in practice, and widely used. In this paper we propose a coarse-grained algorithm, AC2001/3.1, that is worst case optimal and preserves as much as possible the ease of its integration into a solver (no heavy data structure to be maintained during search). Experimental results show that AC2001/3.1 is competitive with the best fine-grained algorithms such as AC-6. The idea behind the new algorithm can immediately be applied to obtain a path consistency algorithm that has the best-known time and space complexity. The same idea is then extended to non-binary constraints.

[*] Preliminary versions of this paper appeared in [BR01, ZY01].

[†] During that work, Christian Bessiere was supported by ILOG under a research collaboration contract ILOG/CNRS/University of Montpelier II, Yuanlin Zhang by the Science Foundation Ireland under Grant 00/PI.1/C075, and Roland Yap and Yuanlin Zhang by the Academic Research Fund, National Univ. of Singapore.

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

1 Introduction

Constraint propagation is a basic operation in constraint programming. It is now well-recognized that its extensive use is necessary when we want to efficiently solve hard constraint satisfaction problems. All the constraint solvers use propagation as a basic step. Thus, each improvement to a constraint propagation algorithm has an immediate effect on the performance of the constraint solving engine. In practical applications, many constraints are of well-known types for which specific algorithms are available. These algorithms generally receive a set of removed values for one of the variables involved in the constraint, and propagate these deletions to the other variables of the constraint. They are usually as cheap as one can expect in cpu time. This state of things implies that most of the existing solving engines are based on a constraint-oriented propagation scheme (ILOG Solver, CHOCO, etc.). We call the algorithms using this scheme *coarse-grained* algorithms. AC-3 [Mac77a, McG79] is a generic constraint propagation algorithm which fits the best this propagation scheme. Its successors, AC-4, AC-6, and AC-7, indeed, were written with a value-oriented propagation where the deletion of a value in the domain of a variable will be propagated only to the affected values in the domains of other variables. Algorithms using this propagation are called *fine-grained* algorithms here. The coarse-grained characteristics of AC-3 explain why it is the algorithm which is usually used to propagate those constraints for which nothing special is known about the semantics (and then for which no specific algorithm is available). When compared to AC-4, AC-6 or AC-7, this algorithm has a second strong advantage, namely, its independence with respect to specific data structure which should be maintained if used during a search procedure. Thus, it has ease of implementation. Fine-grained algorithms, on the other hand, have more complex implementation with possibly higher overheads as there is a need to maintain some complex data structures.

Unfortunately, the worst case time complexity of AC-3 is $\mathcal{O}(ed^3)$, where e is the number of constraints and d is the size of the maximum domain in a problem. Fine grained algorithms on the other hand enjoy optimal worst case complexity $\mathcal{O}(ed^2)$ [MH86]. The fine-grained algorithms are also more efficient when applied to networks where much propagation occurs [BFR95, BFR99] while AC-3 is better when there is very little propagation.

In this paper, we present a new algorithm, AC2001/3.1, which is the first worst case optimal coarse-grained arc consistency algorithm. This result is somewhat surprising since due to the non-optimality result of AC-3 [MF85] from 1985, it is widely held that only fine-grained algorithms have worst case optimality. AC2001/3.1preserves the simplicity of AC-3 while improving on AC-3 in efficiency both in terms of constraint checks and in terms of cpu time. In our experiments, AC2001/3.1 leads to substantial gains over AC-3 both on randomly generated and real-world instances of problems and comparable to AC-6, the fastest fine-grained algorithm.

The idea behind the new algorithm can be applied immediately to obtain a new simple path consistency algorithm, PC2001/3.1, which has the same time and space complexity as the best known theoretical results. We show how to use the same idea for arc consistency on non-binary constraints with a new algorithm, GAC2001/3.1. We also give a detailed comparison of coarse-grained and fine-grained algorithms.

The paper is organized as follows. The preliminaries are given in section 2 before the

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

presentation of AC2001/3.1 and its complexity analysis in section 3. Section 5 extends the idea to path consistency and and generalised arc consistency. Experimental results to benchmarking the performance of the new algorithm with respect to AC-3 and AC-6 are shown in section 4. We compare and contrast most propagation algorithms in section 6 before concluding in section 7.

2 Preliminaries

In this section we give some background material and notations used herein.

Definition 1 *A finite binary* constraint network *(N, D, C) consists of a finite set of variables $N = \{x_1, x_2, \cdots, x_n\}$, a set of domains $D = \{D_1, D_2, \cdots, D_n\}$, where the domain D_i ($i \in 1..n$) is a finite set of values that variable x_i can take, and a set of constraints $C = \{c_1, \cdots, c_e\}$, where each constraint c_k ($k \in 1..e$) is a binary relation on two variables. A constraint on x_i and x_j is usually denoted by c_{ij}. $(a, b) \in c_{ij}$ means that the constraint c_{ij} holds when $x_i = a$ and $x_j = b$. For the problem of interest here, we require that $\forall a, b \;\; a \in D_i, b \in D_j, (a, b) \in c_{ij}$ if and only if $(b, a) \in c_{ji}$. Verifying whether a tuple (a, b) where $a \in D_i$ and $b \in D_j$ is in c_{ij} is called a* constraint check. *A* solution *of a constraint network is an assignment of a value to each variable such that all constraints in the network are satisfied.*

For simplicity, in the above definition we consider only binary constraints, omitting the unary constraint on any variable [Mac77a]. Without loss of generality we assume there is only one constraint between each pair of variables.

Throughout this paper, n denotes the number of variables, d the size of the largest domain, and e the number of constraints in a constraint network. (x_i, a) denotes a value $a \in D_i$.

Definition 2 *Given a constraint network (N, D, C), the* support *of a value $a \in D_i$ under c_{ij} is a value $b \in D_j$ such that $(a, b) \in c_{ij}$. The value a is* viable *with respect to c_{ij} if it has a support in D_j. A constraint c_{ij} is* consistent from x_i to x_j, *that is along the arc (x_i, x_j), if and only if every $a \in D_i$ has a support in D_j. A constraint c_{ij} is* arc consistent *if and only if it is consistent along both arcs (x_i, x_j) and (x_j, x_i). A constraint network is* arc consistent *if and only if every constraint in the network is arc consistent.*

From the definition, we know that a constraint network is arc consistent if and only if every value is viable with respect to every constraint on its variable.

Before presenting and analyzing the new algorithm, let us briefly recall the AC-3 algorithm which is given in Fig. 1 as AC$\mathcal{X}$. The presentation follows [Mac77a, MF85] with a slight change in notation, and node consistency removed. The name of the algorithm AC$\mathcal{X}$ is parameterized by $\mathcal{X}$. For AC-3, the $\mathcal{X}$ is "-3" and thus the procedure REVISE$\mathcal{X}$ is "REVISE-3". For the new algorithm, AC2001/3.1, the $\mathcal{X}$ is "2001/3.1" and thus the procedure REVISE$\mathcal{X}$ is "REVISE2001/3.1".

To enforce arc consistency in a constraint network, a key task of AC-3 is to check the viability of a value with respect to any related constraint. REVISE-3(x_i, x_j) in Fig 2 is to remove those values in D_i without any support in D_j under c_{ij}. If any value in D_i is removed when revising (x_i, x_j), all binary constraints (or arcs) pointing to x_i, except

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

```
algorithm AC𝒳
    begin
1.      Q ← {(x_i, x_j) | c_ij ∈ C or c_ji ∈ C, i ≠ j}
        while Q not empty do
            select and delete any arc (x_i, x_j) from Q
2.          if REVISE𝒳(x_i, x_j) then
3.              Q ← Q ∪ {(x_k, x_i) | c_ki ∈ C, k ≠ j}
    end
```

Figure 1: A schema for coarse-grained arc consistency algorithms

```
procedure REVISE-3(x_i, x_j)
    begin
        DELETE ← false
        for each a ∈ D_i do
1.          if there is no b ∈ D_j such that c_ij(a, b) then
                delete a from D_i
                DELETE ← true
        return DELETE
    end
```

Figure 2: Procedure REVISE for AC-3

c_{ji}, will be revised (line 2 and 3 in Fig 1). A queue Q is used to hold these arcs for later processing. It can be shown that this algorithm is correct.

Proposition 1 ([Mac77a]) *Applying algorithm AC-3 to a constraint network makes it arc consistent.*

The traditional derivation of the complexity of AC-3 is given by the following theorem whose proof from [MF85] is modifi ed in order to facilitate the presentation in Section 3.

Theorem 1 ([MF85]) *Given a constraint network (N, D, C), the time complexity of AC-3 is $\mathcal{O}(ed^3)$.*

Proof. Each arc (x_i, x_j) is revised if and only if it enters Q. The observation is that arc (x_i, x_j) enters Q if and only if some value of D_j is deleted (line 2–3 in Fig 1). So, arc (x_i, x_j) enters Q at most d times and thus is revised d times. Given that the number of arcs is $2e$, REVISE(x_i, x_j) is executed $\mathcal{O}(ed)$ times. The complexity of REVISE(x_i, x_j)

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

in Fig 2 is at most d^2. Hence, the result follows. □

The reader is referred to [Mac77a, MF85] for more details and motivations concerning arc consistency.

Remark. In implementing the queue, to reduce the number of queue operations, one way is simply enqueue the variable whose domain has shrunk, instead of enqueue all relevant arcs. When we dequeue a variable from the queue, we just revise all constraints pointing to this variable. The method is also called variable oriented propagation. This idea appeared in [McG79] and in [CJ98]. In this method, for each variable, one more constraint is revised than in the original algorithm AC-3. However, it seems that the savings from enqueue operations well compensates this cost in our experiments.

3 The New Algorithm

The worst case time complexity of AC-3 is based on a naive implementation of line 1 in Fig. 2 in which b is always searched from scratch. However, from the analysis we know a constraint (x_i, x_j) may be revised many times. The key idea to improve the efficiency of the algorithm is that we need to find from scratch a support for a value $a \in D_i$ *only in the first revision* of the arc (x_i, x_j), and store the support in a structure $Last((x_i, a), x_j)$. When checking the viability of $a \in D_i$ in the subsequent revisions of the arc (x_i, x_j), we only need to check whether its stored support $Last((x_i, a), x_j)$ is still in the domain D_j. If it was removed (because of the revision of other constraints), we would just have to explore the values in D_j that are "after" the support since its "predecessors" have already been checked before.

Assume without loss of generality that each domain D_i is associated with a total ordering $<_d$. The function *succ*(a, D_j), where D_j denotes the current domain of x_j during the procedure of arc consistency enforcing, returns the first value in D_j that is after a in accordance with $<_d$, or NIL, if no such an element exists. We define NIL as a value not belonging to any domain but preceding any value in any domain.

As a simple example, let the constraint c_{ij} be $x_i = x_j$, with $D_i = D_j = [1..11]$. The removal of value 11 from D_j (say, after the revision of some arc leaving x_j) leads to a revision of (x_i, x_j). REVISE-3 will look for a support for every value in D_i, for a total cost of $1 + 2 + \ldots + 9 + 10 + 10 = 65$ constraint checks, whereas only $(x_i, 11)$ had lost support. The new revision procedure makes sure that for each $a \in [1..10]$, $Last((x_i, a), x_j)$ still belongs to D_j, and finds that $Last((x_i, 11), x_j)$ has been removed. Looking for a new support for 11 does not need any constraint check since D_j does not contain any value greater than $Last((x_i, 11), x_j)$, which was equal to 11. It saves 65 constraint checks compared to AC-3.

The new algorithm, AC2001/3.1, is the main algorithm AC$\mathcal{X}$ augmented with the initialization of $Last((x_i, a), x_j)$ to be NIL for any constraint c_{ij} and any value $a \in D_i$. The corresponding revision procedure, REVISE2001/3.1 is given in Fig. 3. In Fig. 3, line 1 checks if the support in $Last$ is still valid and otherwise line 2 makes use of the domain ordering to find the first support after the old one. We now show the correctness of AC2001/3.1.

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

```
procedure REVISE2001/3.1(x_i, x_j)
    begin
        DELETE ← false
        for each a ∈ D_i do
            b ← Last((x_i, a), x_j)
1.          if b ∉ D_j then
                b ← succ(b, D_j)
2.              while (b ≠ NIL) and (¬c_ij(a, b)) do
                    b ← succ(b, D_j)
                if b ≠ NIL then
                    Last((x_i, a), x_j) ← b
                else
                    delete a from D_i
                    DELETE ← true
        return DELETE
    end
```

Figure 3: Procedure REVISE for AC2001/3.1

Theorem 2 *Applying algorithm AC2001/3.1 to a constraint network makes it arc consistent.*

Proof. AC-3 and AC2001/3.1 have exactly the same initialization phase except that AC2001/3.1 stores $Last((x_i, a), x_j)$, the support found for each a on each c_{ij}. It is sufficient to show that REVISE-3(x_i, x_j) and REVISE2001/3.1(x_i, x_j) are equivalent given that D_i and D_j are the same when either procedure is called. In other words, a values is deleted by REVISE-3 iff it is deleted by REVISE2001/3.1. Obviously the return value would also be the same for both procedures. Without loss of generality, we can assume that REVISE-3 visits the same values as the ordering used in $<_d$.

Suppose a value a is deleted by REVISE-3. Line 1 in REVISE-3 tells us that a has no support. Consequently, line 1 in REVISE2001/3.1 is also true and the while loop in line 2 will not find any support. Hence a will be deleted.

Now consider REVISE2001/3.1 deleting a value a. Let b be the previous support, Last$((x_i, a), x_j)$. Since line 1 will be true, b is not a support for a. The while loop at line 2 also doesn't find for a any support after b. Now suppose there is a support b' such that $b' <_d b$. It must also be a support for a in all the previous domains of x_j. Hence, Last$((x_i, a), x_j) \leq_d b'$, which contradicts $b =$ Last$((x_i, a), x_j)$. Thus, a has no support in D_j and will also be deleted by REVISE-3.

Proposition 1 implies that AC2001/3.1 achieves arc consistency on a constraint network. □

Next, we show that AC2001/3.1 has optimal worst case time complexity.

Theorem 3 *The worst case time complexity of AC2001/3.1 is $\mathcal{O}(ed^2)$ with space complexity $\mathcal{O}(ed)$.*

Proof. Here it is helpful to regard the execution of AC2001/3.1 on an instance of a constraint network as a sequence of calls to REVISE2001/3.1(x_i, x_j).

Consider the total time spent on an arc (x_i, x_j). From the proof in Theorem 1, the arc (x_i, x_j) will be revised at most d times.

In the $l^{th}(1 \leq l \leq d)$ revision of (x_i, x_j), let t_l be the time for searching a support for a value $a \in D_i$. t_l can be considered as 1 if $Last((x_i, a), x_j) \in D_j$ (see line 1 in Fig. 3) and otherwise it is s_l which is simply the number of elements in D_j checked after $Last((x_i, a), x_j)$ and before the next support is found (the **while** loop in line 2). So, the total time of the algorithm spent on $a \in D_i$ with respect to (x_i, x_j) is

$$\sum_{1}^{d} t_l \leq \sum_{1}^{d} 1 + \sum_{1}^{d} s_l$$

where $s_l = 0$ if $t_l = 1$. Observe that REVISE2001/3.1(x_i, x_j) checks an element in D_j at most once when looking for a support for $a \in D_i$. Therefore, $\sum_1^d s_l \leq d$ which results in $\sum_1^d t_l \leq 2d$.

To revise (x_i, x_j), we need to find a support for each value of D_i. For there are up to d values in D_i, at most $\mathcal{O}(d^2)$ time will be spent on revising the arc (x_i, x_j).

Hence, the complexity of AC2001/3.1 is $\mathcal{O}(ed^2)$ since the number of arcs in the constraint network is $2e$ (one constraint is regarded as two arcs).

The space complexity of AC2001/3.1 is bounded above by the size of Q, and the structure $Last$. Q can be of complexity in $O(n)$ or $O(e)$, depending on the the implementation of the queue. The size of $Last$ is in $O(ed)$ since each value $a \in D_i$ needs a space in $Last$ with respect to each constraint involving x_i. This gives a $O(ed)$ overall space complexity. □

4 Experimental Results: AC2001/3.1 versus AC-3 and AC-6

We presented AC2001/3.1, a refinement of AC-3 with optimal worst case time complexity. It remains to see whether it is effective in saving constraint checks and/or cpu time when compared to AC-3. As we said previously, the goal is not to compete with AC-6/AC-7, which have very subtle data structure for the propagation phase. However, we will see in the experimental results that AC2001/3.1 is often competitive with these fine-grained algorithms.

There have been many experimental studies on the performance of general arc consistency algorithms [Wal93, Bes94, BFR99]. Here, we take problems used in [BFR99], namely some random CSPs and Radio Link Frequency Assignment Problems (RLFAPs). Given the experimental results of [BFR99], AC-6 is chosen as a representative of a state-of-the-art algorithm because of its good runtime performance over the problems of concern. In addition, a new artificial problem, DOMINO, in the same vein as the problem in Fig. 5 in [DP88], is designed to study the worst case performance of AC-3.

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

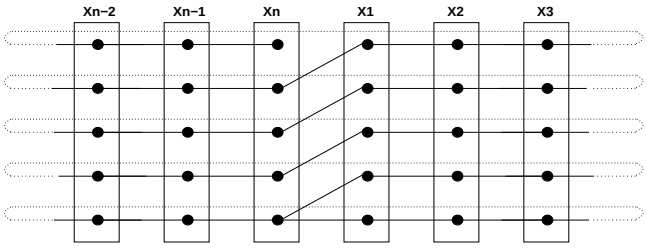


Figure 4: The domino problem

Randomly generated problems. For the random instances, we used a model B generator [Pro96]. The parameters are $\langle N, D, C, T\rangle$, where N is the number of variables, D the size of the domains, C the number of constraints (the *density* $p1$ is equal to $2C/N \cdot (N-1)$), and T the number of forbidden tuples per constraint (the *tightness* $p2$ is equal to T/D^2). We used the generator available in [FBDR96]. For each class of problems tested, we ran the first 50 instances generated using the initial seed 1964 (as in [BFR99]).

RLFAP. The radio link frequency assignment problem (RLFAP) is to assign frequencies to communication links to avoid interference [CdGL$^+$99]. We use the CELAR instances of RLFAP which are real-life problems available in the FullRLFAP archive[1] at `ftp://ftp.cs.unh.edu/pub/csp/archive/code/benchmarks`.

DOMINO. Informally the DOMINO problem is an undirected constraint graph with n variables and a cycle of constraints. The domain of any variable x_i is $D_i = \{1, 2, \ldots, d\}$. The constraints are $C = \{c_{i(i+1)} | \forall i \in 1..n-1\} \cup \{c_{1n}\}$ where $c_{1n} = \{(d, d)\} \cup \{(v, v+1) \mid v < d\}$ is called the *trigger constraint* and the other constraints in C are identity relations. (See the value based constraint graph in Fig. 4.) A DOMINO instance is thus fully characterized by the pair of parameters $\langle n, d\rangle$. The trigger constraint will make one value invalid during arc consistency and that value will trigger the domino effect on the values of all domains until each domain has only one value d left. So, each revision of an arc in coarse-grained algorithms removes one value while fine-grained algorithms only do the necessary work.

Some details of our implementation of AC2001/3.1 and AC-3.0 are as follows. We implemented domains and related operations by double-linked lists. The Q in AC-3 is implemented as a queue with a FIFO policy. For AC-6, we noted that using a single currently supported list per value is faster than using multiple lists with respect to related constraints as needed for AC-7. This may be one reason why AC-7 is slower than AC-6 in [BFR99]. Our implementation of AC-6 adopts a single currently supported list. The code is written in C++ with *g++*. The experiments are run on a PC Pentium II 300MHz processor with Linux. The performance of arc consistency algorithms here is measured along two dimensions: running time and number of constraint checks (#ccks).

[1] We thank the Centre d'Electronique de l'Armement (France).

	AC-3		AC2001/3.1		*AC-6*$^{(*)}$
	#ccks	time	#ccks	time	*time*
$P1$ (under-constrained)	100,010	0.04	100,010	0.05	*0.07*
$P2$ (over-constrained)	507,783	0.18	487,029	0.16	*0.10*
$P3$ (phase transition of AC)	2,860,542	1.06	688,606	0.34	*0.32*
$P4$ (phase transition of AC)	4,925,403	1.78	1,147,084	0.61	*0.66*
SCEN#08 (arc inconsistent)	4,084,987	1.67	2,721,100	1.25	*0.51*

Table 1: Arc consistency results in mean number of constraint checks (#ccks) and mean cpu time in seconds (time). (*) The number of constraint checks performed by AC-6 is similar to that of AC2001/3.1, as discussed in Section 6.

4.1 Arc Consistency as a Preprocessing Step

The first set of experiments shows the efficiency of AC2001/3.1 when arc consistency is used for preprocessing (without search). In this case, the chance to have some propagations is small on real instances. As such, we also choose problems falling in the phase transition of arc consistency (see [GMP$^+$97]). To see the different behaviours, we present results for randomly generated instances with different characteristics (those presented in [BFR99]) and on a RLFAP instance where enforcing arc consistency was not trivial:

- P1= $\langle 150, 50, 500, 1250\rangle$, under-constrained CSPs, where all generated instances are already arc consistent;
- P2= $\langle 150, 50, 500, 2350\rangle$, over-constrained CSPs, where all generated instances are *arc inconsistent*, which means that the instances are not satisfiable and this can be detected by enforcing arc consistency;
- P3=$\langle 150, 50, 500, 2296\rangle$ and P4=$\langle 50, 50, 1225, 2188\rangle$, problems in the phase transition of arc consistency;
- the RLFAP instance SCEN#08, which is arc inconsistent.

Table 1 presents the results. For the randomly generated instances, the number of constraint check (#ccks) and time are averaged over the 50 instances in each class. The under-constrained (P1) and over-constrained (P2) problems. represent cases where there is little or no propagation needed to reach the arc consistent or arc inconsistent state. This is the best case for AC-3. The AC2001/3.1 algorithm still gives comparable runtimes, which indicates that the overhead incurred by AC2001/3.1 is not significant since in P1 there are no savings in constraint checks and P2 only saves about 4% of the checks.

The P3 instances are sparse problems (with a density of 4.5%) at the phase transition of arc consistency. The P4 instances are dense problems (with a complete graph) also at the phase transition of arc consistency. Usually much propagation is needed on these problems to make the network arc consistent. We see that here the runtime of AC2001/3.1 is significantly faster than AC-3 due to large savings in the number of constraint checks.

The final experiment reports the results for a real-life problem, SCEN#08. Here, AC2001/3.1 also saves a significant amount of constraint checks and is also faster.

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

	MAC-3		MAC2001/3.1		*MAC6*
	#ccks	time	#ccks	time	*time*
SCEN#01	5,026,208	2.33	1,983,332	1.62	*2.05*
SCEN#11	77,885,671	39.50	9,369,298	21.96	*14.69*
GRAPH#09	6,269,218	2.95	2,127,598	1.99	*2.41*
GRAPH#10	6,790,702	3.04	2,430,109	1.85	*2.17*
GRAPH#14	5,503,326	2.53	1,840,886	1.66	*1.90*

Table 2: Results for search of the first solution with a MAC algorithm in number of constraint checks (#ccks) and cpu time in seconds (time).

In order to compare AC2001/3.1 to AC-6, it is necessary to first understand that they perform the same number of constraint checks (see Section 6). Here the runtimes show that for most of the problems AC2001/3.1 and AC-6 are comparable and we will return again to this comparison with the DOMINO problem.

4.2 Maintaining Arc Consistency during Search

The second set of experiments we present in this section shows the behaviour of AC2001/3.1 when arc consistency is maintained during search (MAC algorithm [SF94]) to find the first solution. We present results for all the instances contained in the FullRLFAP archive for which more than 2 seconds were needed to find a solution or to prove that none exists. It has to be noticed that the original objective in these instances is to find the "best" solution under some criteria. This is of course out of the scope of this paper.

Table 2 contains the results. From these instances we can see a significant gain for AC2001/3.1 on AC-3, with up to 9 times less constraint checks and twice less cpu time on SCEN#11. As for the experiments performed on random instances at the phase transition of arc consistency, this tends to show that the trick of storing the $Last$ data structure significantly pays off. In addition, we see that in spite of its simple data structures, AC2001/3.1 is faster than AC-6 on all instances except the difficult SCEN#11. The reason why AC-6 takes more time can be explained as follows. The main contribution to the slow down of AC-6 is the maintenance of the currently supported list for each value of each variable. Our experiments show that the overhead of maintaining the list does not usually compensate for the savings, at least under the assumption that constraint checking is cheap.

4.3 The DOMINO Problem

The last set of experiments we made shows the extreme case where the arc consistency process converges after a long propagation that removes all values in all domains but those belonging to the unique solution. The DOMINO problem is designed to exhibit this behaviour. What we can expect from such a pathological case is to show us the deep properties of non optimal coarse-grained, optimal coarse-grained and fine-grained algorithms.

The results are in Table 3. We can see clearly the effect of the non-optimal worst case complexity of AC-3. The number of constraint checks and cpu time increase dramatically

	AC-3		AC2001/3.1		*AC6*
	#ccks	time	#ccks	time	*time*
$\langle 1000, 10\rangle$	319,964	0.19	155,009	0.13	*0.16*
$\langle 500, 100\rangle$	90,845,149	25.70	7,525,099	3.18	*2.66*
$\langle 300, 300\rangle$	1,390,485,449	381.25	40,545,299	15.40	*12.16*

Table 3: Results on the DOMINO problem in number of constraint checks (#ccks) and cpu time in seconds (time).

	AC2001/3.1 domain checks	*AC6* list checks
$\langle 1000, 10\rangle$	53,991	17,999
$\langle 500, 100\rangle$	2,524,401	88,999
$\langle 300, 300\rangle$	13,544,401	179,399

Table 4: Results on the DOMINO problem in the number of domain versus list checks

with the size of the domains. As we already saw, AC2001/3.1 and AC-6 perform exactly the same number of constraint checks. However, as in the most difficult problem of Section 4.2, the fine-grained feature of AC-6 pays off with respect to AC2001/3.1 especially when the domain size increases. This can be explained by the way AC2001/3.1 and AC-6 propagate the deletions of values. If we look more closely at the operations performed by these two algorithms when a value (x_j, b) is deleted, we note that they achieve optimality in two different ways. For each (x_i, a) such that x_i shares a constraint with x_j, AC2001/3.1 checks $Last((x_i, a), x_j)$ against the domain D_j to know whether (x_i, a) still has support (see line 1 in Fig. 3). The $Last$ indicates where to start the new search for support. AC-6 on the other hand, maintains for each (x_j, b) the list of the values a for x_i with $Last((x_i, a), x_j) = b$. When b is deleted from D_j, thanks to these lists of supported values, AC-6 directly knows which values in D_i need to seek a new support, and where to start the new search.

By counting the number of such operations they perform (membership test of a $Last$ in a domain for AC2001/3.1 and list operations on supported lists for AC-6) we obtain the following interesting information. While they don't perform any such tests during the initialization phase, the number of tests they perform during the propagation as shown in Table 4 differs quite significantly. As domain size increases (and thus propagation becomes longer), the cost of AC2001/3.1 propagation increases faster than that of AC-6.

5 An Application to Path Consistency and Non-binary Constraints

5.1 Path Consistency

Notation. In this subsection, to simplify the presentation a variable x_i is represented by its index i.

Assume there is a constraint between any pair of variables in a given constraint network (N, D, C). If it is not the case, we add a special constraint between the uncon-

strained pairs of variables. This constraint allows the constrained variables to take any values. The network is *path consistent* if and only if for any $c_{ij} \in C$, any tuple $(a, b) \in c_{ij}$, and any variable $k \in N$, there exists a value $v \in D_k$ such that the values a, b, and v satisfy the constraints among variables i, j, and k.

The same idea behind AC2001/3.1 applies here. Specifically, in order to find a new support for each $(a, b) \in c_{ij}$ with respect to a variable, say k, it is not necessary to start from scratch every time. We can start from where we stopped before. $Last((i, a), (j, b), k)$ is used to remember that point.

The path consistency algorithm, which we have named PC2001/3.1, partially motivated by the algorithm in [CJ98], is shown in Fig 5. It includes two parts: initialization (INITIALIZE(Q)) and propagation (the **while** loop on Q). During the initialization, a first support is searched for each pair of values $((i, a), (j, b))$ on each third variable k. This support is stored in $Last((i, a), (j, b), k)$. When a tuple (a, b) is removed from c_{ij}, we enqueue $((i, a), j)$ and $((j, b), i)$ into Q. Later, when $((i, a), k)$ is popped from Q, REVISE_PATH($(i, a), k, Q$) (in Fig 6) will check every constraint c_{ij} where $j \in N - \{i, k\}$ to see if any tuple in c_{ij} is affected by the modification of c_{ik}. For each constraint c_{ij}, REVISE_PATH tries to find in D_k a support not from scratch but from its support in the previous revision (line 1 and line 2 in Fig 6) for only those tuples starting with a.

---

algorithm PC2001/3.1
  begin
    INITIALIZE(Q)
    while Q not empty **do**
      Select and delete any $((i, a), j)$ from Q
      REVISE_PATH($(i, a), j, Q$))
    endwhile
  end
procedure INITIALIZE(Q)
  begin
    for any $i, j, k \in N$ **do**
      for any $a \in D_i, b \in D_j$ such that $(a, b) \in c_{ij}$ **do**
        if there is no $v \in D_k$ such that $(a, v) \in c_{ik} \wedge (v, b) \in c_{kj}$
        then
          $c_{ij}(a, b) \leftarrow$ **false**; $c_{ji}(b, a) \leftarrow$ **false**
          $Q \leftarrow Q \cup \{(i, a), j\} \cup \{(j, b), i\}$
        else
          Let $v \in D_k$ be the first value satisfying
            $(a, v) \in c_{ik} \wedge (v, b) \in c_{kj}$
          Last$((i, a), (j, b), k) \leftarrow v$
  end

Figure 5: Algorithm to enforce path consistency

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

```
procedure REVISE_PATH( (i,a), k, Q)
  begin
    for any j ∈ N, j ≠ i, j ≠ k do
      for any b ∈ D_j such that (a,b) ∈ c_ij do
1.      v ← Last((i,a),(j,b),k)
2.      while (v ≠ NIL) ∧ ((a,v) ∉ c_ik ∨ (v,b) ∉ c_kj) do
          v ← succ(v, D_k)
        if v = NIL then
          c_ij(a,b) ← false; c_ji(b,a) ← false
          Q ← Q ∪ {((i,a),j} ∪ {((j,b),i)}
        else Last((i,a),(j,b),k)) ← v
      endfor
  end
```

Figure 6: Revision procedure for PC algorithm

For this algorithm, we have the following result.

Theorem 4 *The time complexity of the algorithm PC2001/3.1 is $\mathcal{O}(n^3d^3)$ with space complexity $\mathcal{O}(n^3d^2)$.*

Proof. The complexity of the algorithm PC depends on the procedure REVISE_PATH whose second loop is to find a support for the tuple $((i,a),(j,b))$ with respect to k. The **while** loop in line 2 (Fig 6) either takes constant time if the condition is not satisfied (the support stored in $Last$ is still valid), or skips values in D_k otherwise. For the second case, no matter how many times we try to find a support for $((i,a),(j,b))$, at most we skip d values since totally we have only d values in D_k.

We know that it is necessary to find a support for $((i,a),(j,b))$ with respect to k if and only if some tuple (a,v) is removed from c_{ik}. So we need to find such a support d times. From first paragraph, for these d times we have at most d constant checks and d skips in total. As a result, to find a support for $((i,a),(j,b))$ with respect to k we need $2d$ steps. Given that i,j,k can be any variable from N and a,b any value from D_i and D_j respectively, we have n^3d^2 possible $((i,a),(j,b))$'s and k's. Hence, the total time cost is $n^3d^2 \times 2d$, that is $\mathcal{O}(n^3d^3)$.

The main working space is for the structure $Last((i,a),(j,b),k)$. The size of this structure is the number of combinations of possible choices for i,j,k,a,b, that is $\mathcal{O}(n^3d^2)$. □

The PC2001/3.1 has time complexity of $\mathcal{O}(n^3d^3)$ and space complexity of $\mathcal{O}(n^3d^2)$ which is the same bounds as the best known results obtained in [Sin96]. The algorithm in [Sin96] employs a supported list for each value of a variable and propagates the removal of values in a fashion of AC-6. Compared with the supported list, the structure

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

$Last((i,a),(j,b),k)$ is easier to maintain. This makes the PC2001/3.1 algorithm both simpler to understand and to implement.

5.2 Non-binary Constraints

AC2001/3.1 can be extended to GAC2001/3.1 to deal with non-binary constraints. The definition of arc consistency for non binary constraints is a direct extension of the binary one [Mac77b, MH86]. Let us denote by $var(c_j) = (x_{j_1}, \ldots, x_{j_q})$ the sequence of variables involved in a constraint c_j, by $rel(c_j)$ the set of tuples allowed by c_j, and by $D^{var(c_j)}_{|x_i=a}$ the set of the tuples τ in $D_{j_1} \times \cdots \times D_{j_q}$ with $\tau[x_i] = a$ (where $i \in \{j_1, .., j_q\}$). A tuple τ in $D^{var(c_j)}_{|x_i=a} \cap rel(c_j)$ is called a support for (x_i, a) on c_j. The constraint c_j is *arc consistent* (also called *generalized arc consistent*, or *GAC*) iff for any variable x_i in $var(c_j)$, every value $a \in D_i$ has a support on c_j. Tuples in a constraint c_j are totally ordered with respect to the lexicographic ordering obtained by combining the ordering $<_d$ of each domain with the ordering of the sequence $var(c_j)$ (or with respect to any total order used when searching for support). Once this ordering is defined, a call to REVISE2001/3.1(x_i, c_j) (see Fig. 7) checks for each $a \in D_i$ whether $Last((x_i, a), c_j)$, which is the smallest support found previously for (x_i, a), still belongs to $D^{var(c_j)}$. If not, it looks for a new support for a on c_j. If such a support τ exists, it is stored as $Last((x_i, a), c_j)$, otherwise a is removed from D_i. The function *succ*$(\tau, D^{var(c_j)}_{|x_i=a})$ returns the smallest tuple in $D^{var(c_j)}_{|x_i=a}$ greater than τ.

---

procedure REVISE2001/3.1(x_i, c_j)
  begin
    DELETE $\leftarrow$ **false**
    for each $a \in D_i$ **do**
      $\tau \leftarrow$ Last$((x_i, a), c_j)$
      if $\exists k / \tau[x_{j_k}] \notin D_{j_k}$ **then**
        $\tau \leftarrow$ *succ*$(\tau, D^{var(c_j)}_{|x_i=a})$
        while $(\tau \neq NIL)$ **and** $(\neg c_j(\tau))$ **do**
          $\tau \leftarrow$ *succ*$(\tau, D^{var(c_j)}_{|x_i=a})$
        if $\tau \neq NIL$ **then**
          Last$((x_i, a), c_j) \leftarrow \tau$
        else
          delete a from D_i
          DELETE $\leftarrow$ **true**
    return DELETE
  end

Figure 7: Procedure REVISE for GAC2001/3.1

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

In Fig. 8, we present a version of the main algorithm based on the one proposed in [Mac77b]. It is a brute force propagation schema that does not take into account the fact that in practice some of the constraints may have *ad hoc* propagators. Thus the algorithm may have to be adapted depending on the architecture of the solver in which it is used. Standard techniques are described in [ILO99, Lab00].

---

algorithm GAC$\mathcal{X}$
  begin
    $Q \leftarrow \{(x_i, c_j) \mid c_j \in C, x_i \in var(c_j)\}$
    while Q not empty **do**
      select and delete any pair (x_i, c_j) from Q
      if REVISE$\mathcal{X}(x_i, c_j)$ **then**
        $Q \leftarrow Q \cup \{(x_k, c_m) \mid c_m \in C, x_i, x_k \in var(c_m), m \neq j, i \neq k\}$
  end

Figure 8: A non binary version of coarse-grained arc consistency algorithm

Complexity. The worst-case time complexity of GAC2001/3.1 depends on the arity of the constraints involved in the constraint network. The greater the number of variables involved in a constraint, the higher the cost to propagate it. Let us first limit our analysis to the cost of enforcing GAC on a single constraint, c_j, of arity $r = |var(c_j)|$. For each variable $x_i \in var(c_j)$, for each value $a \in D_i$, we look for supports in the space $D^{var(c_j)}_{|x_i=a}$, which can contain up to d^{r-1} tuples. If the cost of constraint checks[2] is in $\mathcal{O}(r)$ this gives a cost in $\mathcal{O}(rd^{r-1})$ for checking viability of a value. Since we have to find support for rd values, the cost of enforcing GAC on c_j is in $\mathcal{O}(r^2d^r)$. If we enforce GAC on the whole constraint network, values can be pruned by other constraints, and each time a value is pruned from the domain of a variable involved in c_j, we have to revise c_j. So, c_j can be revised up to rd times. Fortunately, additional calls to REVISE2001/3.1 do not increase its complexity since, as in the binary case, $Last((x_i, a), c_j)$ ensures that the search for support for (x_i, a) on c_j will never check twice the same tuple. Therefore, in a network involving constraints of arity bounded by r, the total time complexity of GAC2001/3.1 is in $\mathcal{O}(er^2d^r)$.

6 Related Work and Discussion

Many arc consistency algorithms have been designed since the birth of the first such algorithm. In this section we present a systematic way to view these algorithms including

[2]The cost of a constraint check is sometimes considered as constant time while it is natural to assume its cost be linear to its arity.

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

AC-3, AC-4, AC-6 , AC-7 and AC2001/3.1. We also present an analysis of the performance of these algorithms, especially AC2001/3.1 and AC-6.

6.1 A Classification and Comparison of AC algorithms

Arc consistency algorithms can be classified by their methods of propagation. So far, two approaches are employed in known efficient algorithms: arc oriented and value oriented. Arc oriented propagation originates from AC-1 and its underlying computation model is the constraint graph where we have only variables and topological relationship between variables derived from constraints.

Definition 3 *The* constraint graph *of a constraint network (N, D, C) is the graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = N$ and $\mathcal{E}$ = $\{(i,j) \mid \exists c_{ij} \in C\}$.*

Value oriented propagation originates from AC-4 and its underlying computation model is the value based constraint graph where each constraint is also represented as a (sub)graph. For example, the graph in Fig 4 is a value based graph where a vertex is a value and an edge is an allowed tuple by the corresponding constraint.

Definition 4 *The* value based constraint graph *of a constraint network (N, D, C) is $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ = $\{(i,a) \mid x_i \in N, a \in D_i\}$ and $\mathcal{E}$ = $\{((i,a),(j,b)) \mid a \in D_i, b \in D_j, c_{ij} \in C, (a,b) \in c_{ij}\}$.*

The value based constraint graph is also known under the names *consistency graph* or *microstructure*. A more specific name for the traditional constraint graph may be *variable based constraint graph*. The key idea of value oriented propagation is that once a value is removed only the viability of those values depending on it will be checked. Thus it is more fine-grained than arc oriented propagation. Algorithms working with variable and value based constraint graphs can be classified respectively as coarse-grained algorithms and fine-grained algorithms.

An immediate observation is that compared with variable based constraint graph, the time complexity analysis in value based constraint graph is straightforward. That is, the total number of operations during the execution of a fine-grained algorithm will be of the same order as the number of edges in the value based constraint graph: $\mathcal{O}(ed^2)$. As far as we know, Perlin [Per92] is the first to make value based constraint graph explicit in arc consistency enforcing algorithm.

Given a computation model of propagation, the algorithms differ in the implementation details. Under variable based constraint graph, AC-3 [Mac77a] can be thought of as an open algorithm, as suggested by our notation AC$\mathcal{X}$. Its time complexity analysis in [MF85] can be regarded as a realized implementation where a very intuitive revision procedure is employed. The new algorithm AC2001/3.1 presented in this paper uses a new implementation of the revision procedure, leading to the optimal worst case time complexity. Our new approach simply remembers the support obtained in the previous revision of an arc while in the old one, the choice is to be lazy, forgetting previous computation. There are also some approaches to improve the space complexity of AC-3 in [McG79, CJ98].

For value based constraint graphs, AC-4 is the first AC implementation and AC-6 is a lazy version of AC-4. AC-7 exploits the bidirectionality on the basis of AC-6. Bidirectionality states that given any c_{ij}, c_{ji}, and any $a \in D_i, b \in D_j$, $(a, b) \in c_{ij}$ if and only if $(b, a) \in c_{ji}$.

Another observation is that the general properties or knowledge of a constraint network can be isolated from a specific arc consistency enforcing algorithm. In fact the idea of *metaknowledge* [BFR99] can be applied to algorithms for either computation model. For example, to save the number of constraint checks, the bidirectionality can be employed also in coarse-grained algorithm, e.g., in [Gas78, LBH03]. Other propagation heuristics [WF92] such as propagating deletion first [BFR99] are also applicable to the algorithms of both models.

We have delineated the AC algorithms which shows that AC2001/3.1 and AC-6 are methodologically different. From a technical perspective, the time complexity analysis of AC2001/3.1 is different from that of AC-6 where the worst case time complexity analysis is straightforward. The point of commonality between AC2001/3.1 and AC-6 is that they face the same problem: the domain may shrink during the process of arc consistency enforcing and thus the recorded support may not be valid in the future. This makes some portions of the implementation of AC2001/3.1 similar to AC-6. We remark that the proof technique in the traditional view of AC-3 does not directly lead to AC2001/3.1 and its complexity results.

6.2 Analysis of the Performance of AC Algorithms

The time complexity of AC-3 is in $\mathcal{O}(ed^3)$ while that of AC-4, AC-6, AC-7 and AC2001/3.1 is in $\mathcal{O}(ed^2)$. As for space complexity, AC-3 uses as little as $\mathcal{O}(e)$ for its queue, AC-4 has a complexity of $\mathcal{O}(ed^2)$, and AC2001/3.1, AC-6 and AC-7 have $\mathcal{O}(ed)$. When dealing with non-binary constraints, GAC3 [Mac77b] has a $\mathcal{O}(er^3d^{r+1})$ time complexity, GAC2001/3.1 is in $\mathcal{O}(er^2d^r)$, while GAC4 [MM88] and GAC-schema [BR97] are in $\mathcal{O}(erd^r)$. GAC4 is a factor r better than GAC2001/3.1 because it computes the d^r possible constraint checks on a constraint once and for all at the beginning, storing the information in lists of supported values. For GAC-schema, the reason is that the use of *multidirectionality* (i.e., bidirectionality for non-binary constraints) prevents it from checking a tuple once for each value composing it.

AC-4 does not perform well in practice [Wal93, BFR99] because it *reaches* the worst case complexity both theoretically and in actual problem instances when constructing the value based constraint graph for the instance. Other algorithms like AC-3 and AC-6 can take advantage of some instances where the worst case doesn't occur. In practice, both artificial and real life problems rarely make algorithms behave in the worst case except for AC-4.[3]

The number of constraint checks is also used to evaluate practical time efficiency of AC algorithms. In theory, applying bidirectionality to all algorithms will result in better performance since it decreases the number of constraint checks. However, if the cost of constraint checks is cheap, the overhead of using bidirectionality may not be compensated by its savings as demonstrated by [BFR99].

[3]However, the value based constraint graph induced from AC-4 provides a convenient and accurate tool for studying arc consistency.

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

AC-6 and AC2001/3.1 have the same worst-case time and space complexities. So, an interesting question here is "What are the differences between AC2001/3.1 and AC-6 in terms of constraint checks?".

Let us first briefly recall the AC-6 behavior [Bes94]. AC-6 looks for one support (the *first* one or *smallest* one with respect to the ordering $<_d$) for each value (x_i, a) with respect to each constraint c_{ij} to prove that a is currently viable. When (x_j, b) is found as the smallest support for (x_i, a) wrt c_{ij}, (x_i, a) is added to $S[x_j, b]$, the list of values currently having (x_j, b) as their smallest support. If (x_j, b) is removed from D_j, it is added to the $DeletionSet$, which is the stream driving propagations in AC-6. When (x_j, b) is picked from the $DeletionSet$, AC-6 looks for the *next* support, greater than b, in D_j for each value (x_i, a) in $S[x_j, b]$.

To allow a closer comparison, we will suppose in the following that the $S[x_i, a]$ lists used in AC-6 are split with respect to each constraint c_{ij} involving x_i, leading to a structure $S[x_i, a, x_j]$, as in AC-7.

Property 1 *Given a constraint network (N, D, C). If we suppose AC2001/3.1 and AC-6 follow the same ordering of variables and values when looking for supports and propagating deletions, then, enforcing arc consistency on the network with AC2001/3.1 performs the same constraint checks as with AC-6.*

Proof. Since they follow the same ordering, both algorithms perform the same constraint checks in the initialization phase: they stop search for support for a value (x_i, a) on c_{ij} as soon as the first $b \in D_j$ compatible with a is found, or when D_j is exhausted (then removing a from D_i). During the propagation phase, both algorithms look for a new support for a value (x_i, a) with respect to c_{ij} only when a value b removed from D_j was the current support for a (i.e., $a \in S[x_j, b, x_i]$ for AC-6, and $b = Last((x_i, a), x_j)$ for AC2001/3.1). Both algorithms search in D_j for a new support for a immediately greater than b. Thus, they will find the same new support for a with respect to c_{ij}, or will remove a, at the same time, and with the same constraint checks. And so on. □

From property 1, we see that the difference between AC2001/3.1 and AC-6 cannot be characterized by the number of constraint checks they perform. We will then focus on the way they find which values should look for a new support. For that, both algorithms handle their specific data structure. Let us characterize the number of times each of them checks its own data structure when a set $\Delta(x_j)$ of deletions from D_j is propagated with respect to a given constraint c_{ij}.

Property 2 *Let c_{ij} be a constraint in a constraint network (N, D, C). Let $\Delta(x_j)$ be a set of values removed from D_j that have to be propagated with respect to c_{ij}. If,*

- $d_A = |\Delta(x_j)| + \sum_{b \in \Delta(x_j)} |S[x_j, b, x_i]|$,
- $d_B = |D_i|$, *and*
- d_C = *# constraint checks performed on c_{ij} to propagate $\Delta(x_j)$,*

then, $d_A + d_C$ and $d_B + d_C$ represent the number of operations AC-6 and AC2001/3.1 will respectively perform to propagate $\Delta(x_j)$ on c_{ij}.

C. Bessière, J-C. Régin, R.H.C. Yap, Y. Zhang: "An Optimal Coarse-grained Arc Consistency Algorithm", Artificial Intelligence, vol 165 (2), pp 165--185, 2005

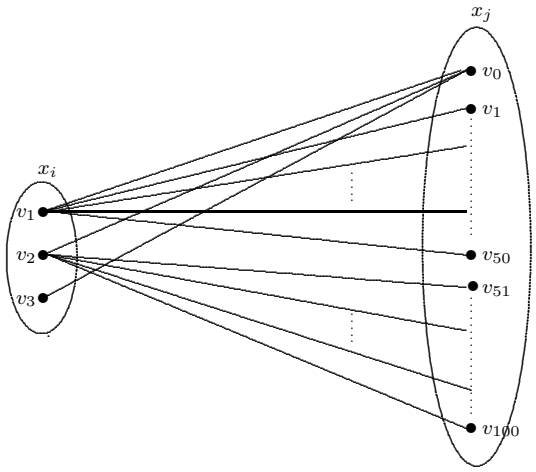


Figure 9: The constraint example

Proof. From property 1 we know that AC-6 and AC2001/3.1 perform the same constraint checks. The difference is in the process leading to them. AC-6 traverses the $S[x_j, b, x_i]$ list for each $b \in \Delta(x_j)$ (i.e., d_A operations), and AC2001/3.1 checks whether $Last((x_i, a), x_j)$ belongs to D_j for every a in D_i (i.e., d_B operations). □

We illustrate this on the extreme case presented in Fig. 9. In that example, the three values of x_i are all compatible with the first value v_0 of x_j. In addition, (x_i, v_1) is compatible with all the values of x_j from v_1 to v_{50}, and (x_i, v_2) with all the values of x_j from v_{51} to v_{100}. Imagine that for some reason, the value v_3 has been removed from D_i (i.e., $\Delta(x_i) = \{v_3\}$). This leads to $d_A = 1$, $d_B = 101$, and $d_C = 0$, which is a case in which propagating with AC-6 is much better than with AC2001/3.1, even if none of them needs any constraint check. Indeed, AC-6 just checks that $S[x_i, v_3, x_j]$ is empty,[4] and stops. AC2001/3.1 takes one by one the 101 values b of D_j to check that their $Last((x_j, b), x_i)$ is not in $\Delta(x_i)$. Imagine now that instead of the value v_3 of D_i these are the values v_1 to v_{100} of D_j that have been removed (i.e., $\Delta_j = \{v_1, \ldots, v_{100}\}$). Now, $d_A = 100$, $d_B = 3$, and $d_C = 0$. This means that AC2001/3.1 will clearly outperform AC-6. Indeed, AC-6 will check for all the 100 values b in $\Delta(x_j)$ that $S[x_j, b, x_i]$ is empty,[5] while AC2001/3.1 just checks that $Last((x_i, a), x_j)$ is not in $\Delta(x_j)$ for every value (totally 3) $a \in D_i$.

Finally, given that both variable and value based constraint graphs can lead to worst case optimal algorithms, we consider their strength on some special constraints: functional, monotonic and anti-functional. For more details, see [VDT92] and [ZY00]. Coarse grained algorithms can be easily adapted to process *monotonic* and *anti-monotonic* constraints in a time complexity of $\mathcal{O}(ed)$ (e.g., using AC2001/3.1). Fine grained algorithms

[4]The only value compatible with (x_i, v_3) is (x_j, v_0), which is currently supported by (x_i, v_1).

[5]Indeed, (x_j, v_0) is the current support for the three values in D_i since it is the smallest in D_j and it is compatible with every value in D_i.

(e.g., AC-4 and AC-6) can deal with *functional* constraints efficiently with complexity $\mathcal{O}(ed)$. We remark that the particular distance constraints in RLFAP can be enforced to be arc consistent in $\mathcal{O}(ed)$ by using a coarse-grained algorithm. It is difficult for coarse-grained algorithm to deal with functional constraints and tricky for fine grained algorithms to handle monotonic constraints. That is why AC-5 [VDT92] is introduced. In fact AC-5 uses both graphs.

By showing that coarse-grained algorithms can be made worst case optimal, this paper opens opportunities to construct new efficient algorithms through reexamining in the context of coarse-grained algorithms those techniques (e.g., bidirectionality and other heuristics or meta knowledge) mainly employed in fine-grained algorithms.

Detailed experiments in [Wal93] show the advantage of AC-3 over AC-4. Our work complements this by providing a way to make coarse-grained algorithms to be worst case optimal.

7 Conclusion

This paper presents AC2001/3.1, a coarse-grained algorithm that improves AC-3. AC2001/3.1 uses an additional data structure, the $Last$ supports, which should be maintained during propagation. This data structure permits a significant improvement on AC-3, and decreases the worst case time complexity to the optimal $O(ed^2)$. AC2001/3.1 is the first algorithm in the literature achieving optimally arc consistency while being free of any lists of supported values. While worst case time complexity gives us the upper bound on the time complexity, in practice, the running time and number of constraint checks are the prime consideration. Our experiments show that AC2001/3.1 significantly reduces the number of constraint checks and the running time of AC-3 on hard arc consistency problems. Furthermore, the running time of AC2001/3.1 is competitive with the best known algorithms, based on the benchmarks from the experiments in [BFR99]. Its behavior is analysed, and compared to that of AC-6, making a contribution to the understanding of the different AC algorithms. The paper shows how the technique used in AC2001/3.1 directly applies to non binary constraints. In addition, this technique can also be used to produce a new algorithm for path consistency. We conjecture from the results of [CJ98] that this algorithm can give a practical implementation for path consistency.

Acknowledgments

The first author would like to thank Philippe Charman who pointed out to him the negative side of fine-grained algorithms. The first author also wants to thank all the members of the OCRE team for the discussions they had about the specification of the CHOCO language.

References

[Bes94] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994.

[BFR95] C. Bessière, E. C. Freuder, and J. C. Régin. Using inference to reduce arc consistency computation. In *Proceedings IJCAI'95*, pages 592–598, Montréal, Canada, 1995.

[BFR99] C. Bessière, E.C. Freuder, and J.C. Régin. Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, 107:125–148, 1999.

[BR97] C. Bessière and J.C. Régin. Arc consistency for general constraint networks: preliminary results. In *Proceedings IJCAI'97*, pages 398–404, Nagoya, Japan, 1997.

[BR01] C. Bessière and J.C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings IJCAI'01*, pages 309–315, Seattle WA, 2001.

[CdGL+99] C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints*, 4:79–89, 1999.

[CJ98] A. Chmeiss and P. Jégou. Efficient path-consistency propagation. *International Journal on Artificial Intelligence Tools*, 7(2):121–142, 1998.

[DP88] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.

[FBDR96] D. Frost, C. Bessière, R. Dechter, and J.C. Régin. Random uniform csp generators. URL: http://www.ics.uci.edu/~dfrost/csp/generator.html, 1996.

[Gas78] J. Gaschnig. Experimental case studies of backtrack vs waltz-type vs new algorithms for satisficing assignment problems. In *Proceedings CCSCSI'78*, pages 268–277, 1978.

[GMP+97] I.P. Gent, E. MacIntyre, P. Prosser, P. Shaw, and T. Walsh. The constrainedness of arc consistency. In *Proceedings CP'97*, pages 327–340, Linz, Austria, 1997.

[ILO99] ILOG. *User's manual*. ILOG Solver 4.4, ILOG S.A., 1999.

[Lab00] F. Laburthe. *User's manual*. CHOCO, 0.39 edition, 2000.

[LBH03] C. Lecoutre, F. Boussemart, and F. Hemery. Exploiting multidirectionality in coarse-grained arc consistency algorithms. In *Proceedings CP'03*, pages 480–494, Kinsale, Ireland, 2003.

[Mac77a] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Mac77b] A.K. Mackworth. On reading sketch maps. In *Proceedings IJCAI'77*, pages 598–606, Cambridge MA, 1977.

[McG79] J.J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphism. *Information Science*, 19:229–250, 1979.

[MF85] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[MH86] R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[MM88] R. Mohr and G. Masini. Good old discrete relaxation. In *Proceedings ECAI'88*, pages 651–656, Munchen, FRG, 1988.

[Per92] M. Perlin. Arc consistency for factorable relations. *Artificial Intelligence*, 53:329–342, 1992.

[Pro96] P. Prosser. An empirical study of phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1–2):81–109, 1996.

[SF94] D. Sabin and E. Freuder. Contradicting conventional wisdosdom in constraint satisfaction. In *Proceedings of the Second Workshop on Principles and Practice of Constraint Programming*, pages 10–20, Rosario, Orcas Island, Washington, 1994.

[Sin96] M. Singh. Path consistency revisited. *Int. Journal on Art. Intelligence Tools*, 6(1&2):127–141, 1996.

[VDT92] P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.

[Wal93] R.J. Wallace. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. In *Proceedings IJCAI'93*, pages 239–245, Chambéry, France, 1993.

[WF92] R.J. Wallace and E.C. Freuder. Ordering heuristics for arc consistency algorithms. In *Proceedings Ninth Canadian Conference on Artificial Intelligence*, pages 163–169, Vancouver, Canada, 1992.

[ZY00] Y. Zhang and R.H.C. Yap. Arc consistency on n-ary monotonic and linear constraints. In *Proceedings CP'00*, pages 470–483, Singapore, 2000.

[ZY01] Y. Zhang and R.H.C. Yap. Making AC-3 an optimal algorithm. In *Proceedings IJCAI'01*, pages 316–321, Seattle WA, 2001.

RAIRO Operations Research
Will be set by the publisher

INEQUALITY-SUM : A GLOBAL CONSTRAINT CAPTURING THE OBJECTIVE FUNCTION*

Jean-Charles Régin[1] and Michel Rueher[2]

Abstract. This paper introduces a new method to prune the domains of the variables in constrained optimization problems where the objective function is defined by a sum $y = \Sigma x_i$, and where the integer variables x_i are subject to difference constraints of the form $x_j - x_i \leq c$. An important application area where such problems occur is deterministic scheduling with the *mean flow time* as optimality criteria. This new constraint is also more general than a sum constraint defined on a set of ordered variables. Classical approaches perform a *local* consistency filtering after each reduction of the bound of y. The drawback of these approaches comes from the fact that the constraints are handled independently. We introduce here a *global constraint* that enables to tackle simultaneously the whole constraint system, and thus, yields a more effective pruning of the domains of the x_i when the bounds of y are reduced. An *efficient algorithm*, derived from Dijkstra's shortest path algorithm, is introduced to achieve interval consistency on this global constraint.

Mathematics Subject Classification. 65K05,90C35

1. Introduction

A great part of the success of *constraint programming* techniques in solving combinatorial problems is due to the capabilities of filtering algorithms to prune the search space. Roughly speaking, a filtering algorithm attempts to remove

.

* *A preliminary version of this paper has been published at CP'2000 [9]*

[1] ILOG Les Taissounieres HB2 1681, route des Dolines Sophia Antipolis 06560 Valbonne, France, regin@ilog.fr

[2] Université de Nice–Sophia-Antipolis, Projet COPRIN I3S/CNRS-INRIA-CERMICS, ESSI, 930, route des Colles - B.P. 145 06903 Sophia-Antipolis, France, rueher@essi.fr

values from the domains of all variables occurring in a constraint whenever the domain of one of these variables is modified.

Arc consistency filtering algorithms on binary constraints are very popular but significant gains in performance have also been obtained during recent years with filtering algorithms associated with more complex constraints [13]. These new filtering algorithms work on so–called "global constraints", e.g., cumulative constraint [2], edge-finder algorithm [4], all–diff constraint [11], cardinality constraint [7, 12]. They take into account the relations between the different occurrences of the same variable in a given set of constraints.

In this paper, we introduce a new global constraint that can achieve significant domain pruning in *constrained optimization problems* where the objective function is defined by a sum $y = \Sigma x_i$, and where the integer variables x_i are subject to difference constraints of the form $x_j - x_i \leq c$. Two important applications where such constraint systems occur are *minimizing mean flow time* and *minimizing tardiness* in deterministic scheduling problems. The following presentation of these applications is adapted from [3].

The mean flow time is defined by $\overline{F} = \frac{1}{n}\sum_{j=1}^{n}(C_j - r_j)$ where C_j and r_j are respectively the completion time and the ready time of task J_j. Difference constraints are due to the precedence constraints and the distances between the tasks (and therefore between their completion times). The mean flow time criterion is important from the user's point of view since its minimization yields a minimization of the mean response time and the mean in-process time of the scheduled tasks set.
The mean tardiness is defined by $\overline{D} = \frac{1}{n}\sum_{j=1}^{n}(D_j)$ where $D_j = \max(C_j - d_j, 0)$, and where d_j is the due date of task J_j. Minimizing this criteria is useful when penalty functions are defined in accordance with due dates.
Both problems are NP-hard in most interesting cases [3,6].

Another useful application of this constraint is a sum constraint defined on an ordered set of variables.

Currently, in the constraint programming framework, such optimization problems are tackled by solving a sequence of decision problems: the solution of each decision problem must not only satisfy the initial constraint system but it must also provide a better bound than the best-known solution. In other words, each new decision problem must satisfy an additional constraint specifying that the value of y is better than the current bound. To take advantage of this additional constraint to cut the search space, and thus to avoid redoing almost always the same work for each decision problem, we introduce here a new global constraint.

In the remainder of this section we first detail the motivation of our approach before showing how it works on a short example.

1.1. Motivation

We consider the constrained optimization problem:

Minimize $f(x)$

subject to $p_i(x) \leq 0 \quad (i = 1, \ldots, m)$

$q_i(x) = 0 \quad (i = 1, \ldots, r)$

where f is a scalar function of a vector x of n components, $p_i(x)$ and $q_i(x)$ are functions which may be non-linear. We assume that an initial box D is given (i.e., the domains of x are bounded) and we seek the global minimum of $f(x)$ in D. For the sake of simplicity, we also assume in the rest of the paper that f is a sum of the form $y = \Sigma_{i=1}^{i=n} x_i$ (In Section 6, we discuss a sum of the form $y = \Sigma_{i=1}^{i=n} a_i x_i$. Unless otherwise mentioned, we assume in the rest of this paper that all the variables take integer values. Efficient filtering algorithms are available for sum constraints but in our case these algorithms are weakened by the fact that variables x_i involved in the objective function also occur in many other constraints. Among all these constraints, there is a subset of binary inequalities that only involve variables occurring in the objective function. Such inequalities may correspond to distance constraints as well as to constraints which have been introduced to break down symmetries of the problem to solve.

Note that the binary inequalities and the sum only model a sub–problem of a real application. Additional constraints are required to capture all the restrictions and features. So, what is needed is an *efficient filtering algorithm* for the conjunction of binary inequalities and the sum constraint.

Dechter et al [5] have shown that shortest path algorithms can efficiently tackle such systems of inequalities in temporal constraint networks problems. The purpose of this paper is to introduce a new global constraint, named `IS`, which handles as a single global constraint the sum constraint and a system of binary inequalities. An efficient algorithm —using a shortest path algorithm on a graph of reduced costs— is introduced to achieve interval consistency (see definition 1) on this global constraint. Before going into the details, let us outline the advantages of this approach on a short example.

1.2. An illustrative example

Consider the constraint network $\mathcal{C} = \{C_1 : x_1 + x_2 = y, \quad C_2 : (x_1 \leq x_2 - 1)\}$ where $D(x_1) = [0, 6], D(x_2) = [1, 7]$ and $D(y) = [1, 13]$. Interval $[a, b]$ denotes the set of integer $S = \{k : a \leq k \wedge k \leq b\}$.

Constraint network $\mathcal{C}$ is arc consistent. Now, suppose that $\min(y)$ is set to 6. Arc consistency is unable to achieve any domain pruning. This is due to the fact that arc consistent filtering handles the constraints one by one. Now, let us examine what happens when constraints C_1 and C_2 are handled as a single constraint. To satisfy constraint C_2, the value of x_1 must be strictly less than the value of x_2, and thus, constraint C_1 cannot be satisfied when x_2 takes its values in $[1, 3]$. So, values $[1, 3]$ in $D(x_2)$ can be deleted. On this example, a global handling of C_1 and C_2 drastically reduces the search space.

1.3. A BRIEF SUMMARY OF OUR FRAMEWORK

The filtering process on C_1 and C_2 is exactly what will be performed on global constraint IS. More precisely, let:

- a sum constraint S_{um} defined by $y = \Sigma_{i=1}^{i=n} x_i$,
- a set of binary inequalities $\mathcal{I}_{neq} = \{x_i - x_j \leq c_{ji}, \quad (i,j \in [1,n])\}$,
- a set of domain constraints $\mathcal{D}_{om} = \{l_i \leq x_i \leq u_i, \quad (i \in [1,n])\}$.

The global constraint IS is defined by $\mathcal{I}_{neq} \cup \mathcal{D}_{om} \cup \{S_{um}\}$. Each time a bound of y is modified, the filtering on IS starts by performing the following operations:

(1) Filtering $\{\mathcal{I}_{neq} \cup \mathcal{D}_{om}\}$ by interval consistency;
(2) Filtering S_{um} by interval consistency;
(3) Updating the bounds of every x_i with respect to constraint set $\mathcal{I}_{neq} \cup \mathcal{D}_{om} \cup \{S_{um}\}$.

Step 1 can be achieved with a shortest path algorithm on the graph associated with $\{\mathcal{I}_{neq} \cup \mathcal{D}_{om}\}$. (See Section 3.2.) Since the graph is likely to contain a negative cycle, this step can be achieved in $O(mn)$ running time.
It is also easy to show that step 2 can be performed with a simple algorithm that runs in $O(n)$ where n is the number of variables. (See Section 3.1.)
The contribution of this paper is an efficient *filtering algorithm* for enforcing interval consistency on the global constraint IS.

Outline of the paper: Section 2 introduces the notation and recalls the basics of CSP and of shortest paths that are needed in the rest of the paper. Section 3 successively shows how interval consistency can be achieved on a sum constraint and on binary inequalities. Section 4 defines interval consistency on the global constraint IS while Section 5 details the algorithm for finding a minimum value of x_i with respect to constraints $\mathcal{I}_{neq} \cup \mathcal{D}_{om} \cup \{S_{um}\}$.

2. BACKGROUND

In order to make this paper self-contained, we now introduce the required background of CSP and of shortest paths.

2.1. BASICS OF CSP

A finite **constraint network** $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is defined by :

- a set of *variables* $\mathcal{X} = \{x_1, ..., x_n\}$;
- a set $\mathcal{D} = \{D(x_1), ..., D(x_n)\}$ of current *domains* where $D(x_i)$ is a finite set of possible values for variable x_i;
- a set $\mathcal{C}$ of constraints between the variables.

A total ordering $\prec$ can be defined on the domains without loss of generality. We will denote by $\min(x_i)$ and $\max(x_i)$ the minimal and the maximal value of $D(x_i)$ w.r.t. to $\prec$.
$|\mathcal{C}|$ denotes the number of constraints while $|X|$ denotes the number of variables. A

constraint C on the ordered set of variables $X(C) = (x_1, ..., x_r)$ is a subset $T(C)$ of the Cartesian product $D(x_1) \times ... \times D(x_r)$ that specifies the *allowed* combinations of values for the variables $(x_1, ..., x_r)$. An element of $D(x_1) \times ... \times D(x_r)$ is called a *tuple* on $X(C)$ and is noted τ. $\tau[k]$ is the k^{th} value of τ. $|X(C)|$ is the *arity* of C.

A value a for x is often denoted by (x, a) while *index*(C, x) is the position of x in $X(C)$.

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a constraint network. The tuple $\tau = (v_1, \ldots, v_n)$ is a **solution** of P if the assignment $((x_1, v_1), \ldots, (x_n, v_n))$ satisfies all the constraints of $\mathcal{C}$. A value v is a **feasible value** for x if there exists a solution in which $x = v$. Let C be a constraint of $\mathcal{C}$. A tuple τ of $X(C)$ is *valid* if $\forall (x, a) \in \tau, a \in D(x)$. A value $a \in D(x)$ is *consistent with* C, either if $x \notin X(C)$, or if there exists a valid tuple $\tau \subset T(C)$ such that $a = \tau[\text{index}(C, x)]$. A constraint is **arc consistent** iff $\forall x_i \in X(C), D(x_i) \neq \emptyset$ and $\forall a \in D(x_i)$, a is consistent with C.

Filtering by arc consistency is often too costly for non-binary constraints and global constraints. **Interval consistency** [8] can be achieved more efficiently. Interval consistency is derived from a relaxation of arc consistency for continuous domains. It is based on an approximation of finite domains by finite sets of successive integers. More precisely, if D is a domain, interval consistency works on D^*, the set of integers $\{k : \min(D) \leq k \leq \max(D)\}$ where where $\min(D)$ and $\max(D)$ denote respectively the minimum and maximum values in D. In the following D^* is called an interval of integers and denoted by $[\min(D), \max(D)]$.

A constraint C is interval–consistent if the following properties hold:

(1) For all x_i in $X(C)$, $\min(D(x_i)) \leq \max(D(x_i))$
(2) For all x_i in $X(C)$, C is arc-consistent when $D^*(x_i)$ is restricted to $\{\min(D(x_i)\}$ and $D(x_j)$ is extended to $D^*(x_j)$ for all $i \neq j$
(3) For all x_i in $X(C)$, C is arc-consistent when $D^*(x_i)$ is restricted to $\{\max(D(x_i)\}$ and $D(x_j)$ is extended to $D^*(x_j)$ for all $i \neq j$.

For specific constraint systems, interval consistency and arc consistency are equivalent. In particular, this is the case for constraints $\mathcal{I}_{neq} \cup \mathcal{D}_{om} \cup S_{um}$ if the domains are finite sets of successive of integers (i.e., if $D_0^*(x_i) = D_0(x_i)$ for all variables). However, for more complex constraints this property does not hold. Consider for instance constraint $x^2 = 4$ and $D(x) = [-2, 2]$. This constraint is interval–consistent but not arc–consistent since $(x, 0)$ is not consistent with this constraint.

2.2. Basics of shortest paths

We briefly recall here a few ideas about shortest paths that are needed in the rest of the paper. Most of the definitions are due to Tarjan [14].

Let $G = (X, U)$ be a directed graph, where X is a set of nodes and U a set of arcs; m denotes $|U|$ whereas n denotes $|X|$. Each arc (i, j) is associated with an integer called the cost of the arc and denoted c_{ij}. A *path* from node v_1 to node v_k in G is a list of nodes $[v_1, ..., v_k]$ such that (v_i, v_{i+1}) is an arc for $i \in [1..k-1]$. A path is *simple* if all its nodes are distinct. A path is a *cycle* if $k > 1$ and $v_1 = v_k$.

The *length* of a path p, denoted by $length(p)$, is the sum of the costs of the arcs contained in p. A *shortest path* from a node s to a node t is a path from s to t whose length is minimum. A cycle of negative length is called a *negative cycle*. There is a shortest path from s to t iff no path from s to t contains a negative cycle. $d(u,v)$ denotes the shortest path distance from node u to node v in G while s denotes the source node.
G_{rc} is the graph derived from G by replacing, for each arc (u,v), c_{uv} with its reduced cost $rc_{uv} = c_{uv} + d(s,u) - d(s,v)$. The shortest path distance from node a to node b in G_{rc} is denoted by $d^0(a,b)$. The following properties [1] hold in G_{rc}:

(1) $\forall (u,v) \in G$: $rc_{uv} \geq 0$
(2) $d(a,b) = d^0(a,b) - d(s,a) + d(s,b)$

3. Interval consistency filtering

This section successively shows how interval consistency can be achieved on a sum constraint and on binary inequalities.

3.1. Sum constraint

We will consider the following definition of a sum constraint:

Definition 1. *Let $X = \{x_1, ..., x_r\}$ be a set of variables. $S_{um} = SUM(X,y)$ is a* sum *constraint defined by the set of tuples $T(S_{um})$:*
$T(S_{um}) = \{\tau : \tau$ *is a tuple of* $X \cup \{y\} \ \wedge \ \sum_{i=1}^{r} \tau[i] - \tau[index(S_{um}, y)] = 0\}$

Proposition 1. *Let $X = \{x_1, ..., x_r\}$ be a set of variables whose domain are interval of integers. Then, for every value v such that $\sum_{x_i \in X} min(D(x_i)) \leq v \leq \sum_{x_i \in X} max(D(x_i))$ there exists an instantiation of the variables of X such that $\sum_{x_i \in X} x_i = v$.*

Proof:
Let $Smin = \sum_{x_i \in X} min(D(x_i))$. We have $Smin \leq v \leq \sum_{x_i \in X} max(D(x_i))$. Consider any ordering of the variables of X: $\{x_1, ..., x_r\}$. There exists an index i such that $v = Smin + \sum_{j=1}^{i-1}(max(D(x_j) - min(D(x_j)) + p$ with $p \leq max(D(x_i)) - min(D(x_i))$. Since $D^*(x_i)$ is the interval of integers $[\min(D(x_i)), \max(D(x_i))]$ then $p \in D^*(x_i)$ and the instantiation of X defined by $x_j = max(D(x_j))$ if $j < i$, $x_i = p + min(D(x_i))$, and $x_j = min(D(x_j))$ if $j > i$ satisfies $\sum_{x_i \in X} x_i = v$. $\odot$

From this proposition we have:

Corollary 1. *Let $X = \{x_1, ..., x_r\}$ be a set of variables whose domain are interval of integers. Then establishing the interval consistency of the constraints $\sum_{x_i \in X} x_i \leq v$ and $\sum_{x_i \in X} x_i \geq v$ is equivalent to establish the interval consistency of the constraint $\sum_{x_i \in X} x_i = v$.*

From Corollary 1 we immediatly have:

Proposition 2. *Let $X \cup \{y\}$ be a set of variables and let $S_{um} = SUM(X, y)$ be a sum constraint. S_{um} is interval–consistent if and only if the following four conditions hold:*

(1) $\min(y) \geq \sum_{x_i \in X} \min(x_i)$
(2) $\max(y) \leq \sum_{x_i \in X} \max(x_i)$
(3) $\forall x_i \in X : \min(x_i) \geq \min(y) - \sum_{x_j \in X - \{x_i\}} \max(x_j)$
(4) $\forall x_i \in X : \max(x_i) \leq \max(y) - \sum_{x_j \in X - \{x_i\}} \min(x_j)$

Interval consistency filtering of $SUM(X, y)$ can be achieved efficiently in an incremental way. The essential observation is that $\sum_{x_j \in X - \{x_i\}} \max(x_j)$ is equal to $\sum_{x_j \in X} \max(x_j) - \max(x_i)$ and $\sum_{x_j \in X - \{x_i\}} \min(x_j)$ is equal to $\sum_{x_j \in X} \min(x_j) - \min(x_i)$. Since the sum over X can be computed only once, the above conditions can be checked in $O(n)$. Thus, the cost of updating the intervals after a modification of bounds of several variables is in $O(n)$. What is instructive with this complexity is the fact that it does not depend on the size of the domains of the variables.

We would like to emphasize that if the domains of the variables are not considered as interval of integers it becomes difficult to establish the consistency of the constraint as shown by the following proposition:

Proposition 3. *Finding a tuple on the variables of X such that $\sum_{x_i \in X} x_i = v$ is an NP-Complete problem in general.*

Proof:
This problem is obviously in NP (easy polynomial certificate). We transform SUMSET-SUM to this problem. SUMSET-SUM is: *Instance*: finite set A, size $s(a_i)$ in Z+ for each $a_i \in A$, positive integer k. *Question*: is there a subset A' of A such that the sum of the sizes of the elements in A' is exactly k?
For each $a_i \in A$ we define a variable x_i whose domain is $\{0, s(a_i)\}$. Then $sum_{x_i \in X}(x_i) = k$ exactly solves SUBSET-SUM. $\odot$

3.2. Binary inequalities

Arc consistency can be achieved on binary inequalities like $\mathcal{I}_{neq}$ by using specific filtering algorithms such as AC-5 [15]. However, the complexity of such algorithms depends on the size of domains of the variables. Thus, they are rather ineffective for detecting inconsistencies. Interval consistency can be achieved in $O(mn)$ where n is the number of variables and $m = |\mathcal{I}_{neq}| + 2n$. This is due to a result of Dechter et al. [5] on the "Simple Temporal Constraint Satisfaction Problem"(STCSP). Roughly speaking, interval consistency can be achieved by searching for shortest paths in a particular graph $G = (N, E)$, called the distance graph, where node set N represents the variables and arc set E stands for the inequality constraints.
More formally, let $P = (\mathcal{X}, \mathcal{D}_c^*, \mathcal{I}_{neq})$ be a CSP where $\mathcal{D}_c^*$ denotes a set of continuous domains. The distance graph $G = (N, E)$ associated with P is defined in the following way (see figure 3.2):

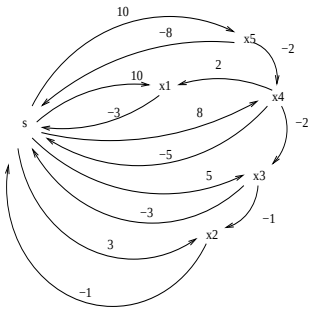


FIGURE 1. Distance graph associated to the CSP $P = \{\{x, y\}\{[1, 6], [2, 5]\}, \{x \leq y - 3\}\}$

- The node set N contains:
  - A special node s, named *source*, with a domain $D(s)$ that is reduced to a single value $\{0\}$.
  - One node for each variable x_i in $\mathcal{X}$.
- The arc set E contains:
  - An arc (x_j, x_i) with cost c_{ji} for each inequality $x_i \leq x_j + c_{ji}$.
  - An arc (x_i, s) with cost $-\min(x_i)$ for each variable x_i in $\mathcal{X}$.
  - An arc (s, x_i) with cost $\max(x_i)$ for each variable x_i in $\mathcal{X}$.

  Arcs (x_i, s) and arcs (s, x_i) result from the definition of domain $D_c^*(x_i) = [\min(x_i), \max(x_i)]$ by the inequalities: $0 \leq x_i - \min(x_i)$ (so $s \leq x_i - \min(x_i)$) and $x_i \leq \max(x_i)$ (so $x_i \leq s + \max(x_i)$).

This problem statement results from the following optimality condition of shortest paths: $d(s, x_j) \leq d(s, x_i) + c_{ij} \quad$ for all $(x_i, x_j) \in N$. This inequality states that for every arc (x_i, x_j) in the network the length of the shortest path to node x_j is not greater than the length of the shortest path to node x_i plus the length of the arc (x_i, x_j). Dechter et al. have shown [5] that :

Theorem 1. *A STCSP is consistent iff its distance graph has no negative directed cycles.*

Theorem 2. *Let G be the directed graph representation of a consistent STCSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ where $\mathcal{C}$ is a set of binary inequalities. The set of feasible values for x_i is $[-d(x_i, s), +d(s, x_i)]$, where $d(x_i, x_j)$ denotes the shortest path from node x_i to node x_j.*

Dechter et al. have extended network based methods for solving mixed-integer linear problems [10]. So, it it is trivial to show that their results hold when the domains are restricted to intervals of integers.

Theorem 1 states that the problem has no solution if G contains a negative cycle. Indeed, a negative cycle indicates that some of the inequalities are contradictory. The following property results from Theorem 2:

Proposition 4. *Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a STCSP and let $G = (N, E)$ be the distance graph associated with $P^* = (\mathcal{X}, \mathcal{D}^*, \mathcal{C})$. If G contains no negative cycle, then $\forall x_i \in \mathcal{X} : (D^*(x_i) = [-d(x_i, s), +d(s, x_i)]) \Rightarrow P$ is interval–consistent*

Proof:
Assume that $D^*(x_i) = [-d(x_i, s), +d(s, x_i)]$. Since G contains no negative cycles, it results from Theorem 2 that $-d(x_i, s)$ and $d(s, x_i)$ are feasible values. Thus, P is interval–consistent. $\odot$

According to Theorem 2, interval consistency can be achieved by computing the shortest paths between s and the x_i, when G does not contain any negative cycle. Computing shortest paths when the problem graph is likely to contain a negative cycle can be achieved in $O(mn)$ running time [14]. When the graph contains no negative arcs, Dijkstra's algorithm computes shortest paths in $O(m + n \log n)$. Of course, Dijkstra's algorithm can always be used on the graph of reduced costs. A nice property of the distance graph $G = (N, E)$ associated with $P^* = (\mathcal{X}, \mathcal{D}^*, \mathcal{C})$ is that the reduced costs can be derived from the minimal and maximal values of the domains.

Proposition 5. *Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP and let $G = (N, E)$ be the distance graph associated with $P^* = (\mathcal{X}, \mathcal{D}^*, \mathcal{C})$. If P is interval–consistent, then the following relations hold:*

$\forall x_i, x_j \in \mathcal{X} : \quad rc_{ij} = c_{ij} + \max(x_i) - \max(x_j)$
$\forall x_i, x_j \in \mathcal{X} : \quad d(x_i, x_j) = d^0(x_i, x_j) - \max(x_i) + \max(x_j)$

These properties trivially result from the definition of the reduced costs and Theorem 2.

4. Global IS constraint

Now, let us show how interval consistency of the global constraint `IS` can be achieved. A global constraint `IS` represents the conjunction of a sum constraint and a set of binary inequalities defined on variables involved in the sum constraint. More formally, we have:

Definition 2. *Let $SUM(X, y)$ be a sum constraint, and $\mathcal{I}_{neq}$ be a set of binary inequalities defined on $X = (x_1, \ldots, x_r)$. Global constraint $\texttt{IS}(X, y, \mathcal{I}_{neq})$ is defined by the set of tuples $T(IS)$:*

$T(IS) = \{\tau : \tau$ *is a tuple of* $X(IS) \ \wedge \ \sum_{i=1}^{r} \tau[i] - \tau[index(IS, y)] = 0 \ \wedge$
$\forall (x_i \leq x_j + c_{ji}) \in \mathcal{I}_{neq} : \tau[i] \leq \tau[j] + c_{ji} \ \}$

To define interval consistency for `IS`, we have to extend inequalities of Proposition 2 in order to take into account the binary inequalities between the variables

involved in the sum constraint.

Let us highlight this point by considering again the initial example. Now, the constraint network is expressed with one global constraint IS:

$\texttt{IS}(\{x_1, x_2\}, y, \{(x_1 \leq x_2 - 1)\})$
where $D(x_1) = [0, 6]$, $D(x_2) = [1, 7]$ and $D(y) = [1, 13]$.

Suppose that $\min(y)$ is set to 6. Inequality (3) of Proposition 2 states that:

$\forall x_i \in X : \min(x_i) \geq \min(y) - \sum_{x_j \in X - \{x_i\}} \max(x_j)$.

So, for x_2, we have: $\min(x_2) \geq 6 - \max(x_1)$. Since $\max(x_1) = 6$. This inequality holds when $\min(x_2)$ is equal to 1 although the constraint $(x_1 \leq x_2 - 1)$ is violated for $x_2 = 1$ and $x_1 = 6$.

Thus, inequality (3) must be modified in order to take into account the constraint $(x_1 \leq x_2 - 1)$. More precisely, the value of x_i and the upper bounds of x_j considered in inequality (3) should satisfy the constraints $(x_i \leq x_j - c_{ji})$ of $\mathcal{I}_{neq}$.

Let $\min_{(x_i \leftarrow a)}(x_j)$ and $\max_{(x_i \leftarrow a)}(x_j)$ respectively be the minimum and the maximum values of $D^*(x_j)$ which satisfy the binary inequalities $\mathcal{I}_{neq}$ when x_i is instantiated to a. Using this notation, inequality (3) can be rewritten in the following form:

$\forall x_i \in X : (x_i \leftarrow a) \Rightarrow a \geq \min(y) - \sum_{j \neq i} \max_{(x_i \leftarrow a)}(x_j)$

This new inequality does not hold for $x_2 = 1$ and $x_1 = 6$. The smallest value of $D^*(x_2)$ that satisfies this inequality is 4.

So, interval-consistency of IS can be defined in the following way :

Proposition 6. *Let $X \cup \{y\}$ be a set variables and let $\texttt{IS}(X, y, \mathcal{I}_{neq})$ be a global sum constraint. IS is interval-consistent iff the following conditions hold:*

(1b) $\min(y) \geq \sum_{x_i \in X} \min(x_i)$
(2b) $\max(y) \leq \sum_{x_i \in X} \max(x_i)$
(3b) $\forall x_i \in X : \min(x_i) \geq \min(y) - \sum_{x_j \in X - \{x_i\}} \max_{(x_i \leftarrow \min(x_i))}(x_j)$
(4b) $\forall x_i \in X : \max(x_i) \leq \max(y) - \sum_{x_j \in X - \{x_i\}} \min_{(x_i \leftarrow \max(x_i))}(x_j)$

Proof:
From inequalities (1b) and (2b) it results that constraint $SUM(X, y)$ holds when y is set to $\min(y)$ and when y is set to $\max(y)$. Since y occurs only in $SUM(X, y)$, it follows that IS is interval-consistent for y. Inequality (3b) ensures that both constraint $SUM(X, y)$ and the inequalities of $\mathcal{I}_{neq}$ hold when x_i is set to $\min(x_i)$. This reasoning remains valid for inequality (4b).
Conversely, it is trivial to show that inequalities (1b), (2b), (3b), and (4b) hold if IS is interval-consistent. ⊙

The scheme of the interval consistency filtering algorithm of constraint IS is given in Algorithm 1. This algorithm is started whenever one bound of a variable in $\mathcal{X} \cup \{y\}$ are modified. Note that steps 1 and 2 are systematically performed when interval consistency on IS is achieved for the first time. Negative cycles are

detected in step 1 of Algorithm 1.
In the rest of this paper, we will only detail the search process of the minimum value of a variable (step 3 in Algorithm 1) since the same kind of reasoning holds for searching maximum values (step 4 in Algorithm 1).

Algorithm 1: Filtering IS by interval consistency

(1) Interval consitency is achieved on $SUM(X, y)$ with an algorithm derived from Proposition 2 whenever a bound of y is modified.
(2) Interval consitency is achieved on the conjunction of binary inequalities $\mathcal{I}_{neq} \cup \mathcal{D}_{om}$ with a shortest path algorithm whenever a bound of some variable of X is modified.
(3) For every variable $x \in X$, the minimum value of x satisfying inequality (3b) will be computed;
(4) For every variable $x \in X$, the maximum value of x satisfying inequality (4b) will be computed.

5. Computing a new minimum

The goal is to find the smallest value $\underline{x_i} \in [\min(x_i), \max(x_i)]$ such that inequality (3b) of Proposition 6 holds. It follows from inequality (3b) that

$$\underline{x_i} = \min(y) - \sum_{x_j \in X - \{x_i\}} \max_{(x_i \leftarrow \underline{x_i})}(x_j) \quad (1)$$

So, we have to determine the largest value of x_j which is consistent with $\underline{x_i}$.
The algorithm is based on the following properties:

Proposition 7. *Let $d_{G_{x_i \leftarrow \underline{x_i}}}(s, x_j)$ denote the shortest path from s to x_j in the graph derived from G by instantiating x_i with $\underline{x_i}$. Then,*

$$\max_{(x_i \leftarrow \underline{x_i})}(x_j) = d_{G_{x_i \leftarrow \underline{x_i}}}(s, x_j)$$

Proof: This property results from Theorem 2 when $G_{x_i \leftarrow \underline{x_i}}$ does not contain any negative cycle. Negative cycles are detected in step 1 of Algorithm 1. So, if there exists a negative cycle, it has been introduced by the instantiation of x_i with $\underline{x_i}$ in $G_{x_i \leftarrow \underline{x_i}}$.
Suppose that the instantiation of x_i with $\underline{x_i}$ introduces a negative cycle in $G_{x_i \leftarrow \underline{x_i}}$. This cycle would contain x_i, so we would have $d(s, x_i) + d(x_i, s) < 0$ and thus $\underline{x_i} - \min(x_i) < 0$ which would be in contradiction with $x_i \in [\min(x_i), \max(x_i)]$ $\odot$

To compute $d_{G_{x_i \leftarrow \underline{x_i}}}(s, x_j)$ efficiently, we have to establish a link between the shortest paths in G and the ones in $G_{x_i \leftarrow \underline{x_i}}$; otherwise we would have to compute the shortest paths at each step on a new graph. The following propositions help us to establish such a connection.

Proposition 8. *Let* $G = (X, U)$.
$\forall s, x_i, x_j \in X : d_G(s, x_j) = \min\left(d_G(s, x_i) + d_{G-\{s\}}(x_i, x_j), d_{G-\{x_i\}}(s, x_j)\right)$

Proof: First, let us recall that in a graph without negative cycles, it exists a simple shortest path whenever a shortest path exists. So, we have to examine two cases:

(1) No simple shortest path from s to x_j contains x_i. Then, we have $d_G(s, x_j) = d_{G-\{x_i\}}(s, x_j)$ and the proposition holds.
(2) Some simple shortest path P from s to x_j contains x_i. That's to say $P = d_G(s, x_i) + d_G(x_i, s)$. Since $d_G(s, x_i)$ contains x_i, s cannot belong to $d_G(x_i, s)$ and we have $d_G(s, x_j) = d_G(s, x_i) + d_{G-\{s\}}(x_i, x_j)$. ⊙

Next property states that if the shortest path from s to x_j goes through x_i when x_i is instantiated to $\underline{x_i}$, then $\max_{(x_i \leftarrow \underline{x_i})}(x_j) = \underline{x_i} + d_{G-\{s\}}(x_i, x_j)$, otherwise $\max_{(x_i \leftarrow \underline{x_i})}(x_j) = \max(x_j)$.

Proposition 9. *After the achievement of steps 1 and 2 of Algorithm 1 we have:*
$\max_{(x_i \leftarrow \underline{x_i})}(x_j) = \min\left(\underline{x_i} + d_{G-\{s\}}(x_i, x_j), \max(x_j)\right)$

Proof: Proposition 8 states for the graph $G_{x_i \leftarrow \underline{x_i}}$:
$d_{G_{x_i \leftarrow \underline{x_i}}}(s, x_j) = \min\left(d_{G_{x_i \leftarrow \underline{x_i}}}(s, x_i) + d_{G_{x_i \leftarrow \underline{x_i}}-\{s\}}(x_i, x_j), d_{G_{x_i \leftarrow \underline{x_i}}-\{x_i\}}(s, x_j)\right)$
Since the instantiation of x_i to $\underline{x_i}$ only modifies the values of c_{sx_i} and of $c_{x_i s}$, the shortest paths in $G - \{s\}$ are equal to the shortest paths in $G_{x_i \leftarrow \underline{x_i}} - \{s\}$. So, we have :
$d_{G_{x_i \leftarrow \underline{x_i}}}(s, x_j) = \min\left(d_{G_{x_i \leftarrow \underline{x_i}}}(s, x_i) + d_{G-\{s\}}(x_i, x_j), d_{G-\{x_i\}}(s, x_j)\right)$.
On the other hand, after the achievement of step 2 of Algorithm 1, Theorem 2 ensures that the set of feasible values for x_i is $[-d(x_i, s), +d(s, x_i)]$. The graph $G_{x_1 \leftarrow \underline{x_i}}$ is the graph G where the values of c_{sx_i} and of $c_{x_i s}$ have respectively been set to $\underline{x_i}$ and to $-\underline{x_i}$. Since $\underline{x_i} \in [-d(x_i, s), +d(s, x_i)]$, $\underline{x_i}$ is a valid value, and thus, Theorem 2 on $G_{x_1 \leftarrow \underline{x_i}}$ states that the set of feasible values of x_i is $[-d_{G_{x_1 \leftarrow \underline{x_i}}}(x_i, s), +d_{G_{x_1 \leftarrow \underline{x_i}}}(s, x_i)] = [\underline{x_i}, \underline{x_i}]$. So, $d_{G_{x_1 \leftarrow \underline{x_i}}}(s, x_i) = \underline{x_i}$.
Moreover, from Theorem 2 we have $d(s, x_j) = max(x_j)$. Therefore the proposition holds. ⊙

Let S be the set of variables x_j for which $\underline{x_i} + d_{G-\{s\}}(x_i, x_j) > \max(x_j)$, that's to say $d_{G-\{s\}}(x_i, x_j) > \max(x_j) - \underline{x_i} = c_{x_i,s} + d(s, x_j) = d(x_i, x_j)$. In other words, S is the set of variables for which the shortest path $d_{G-\{s\}}(x_i, x_j)$ in $G - \{s\}$ is greater than the shortest path $d(x_i, x_j)$ in G.

Then, by Proposition 9, Equation 1 is equivalent to:

$$\underline{x_i} = \min(y) - \sum_{x_k \in S} \max(x_k) - \sum_{x_j \in X-(S \cup \{x_i\})} \underline{x_i} + d_{G-\{s\}}(x_i, x_j) \qquad (2)$$

which is equivalent to

$$\underline{x_i} = \left\lceil \frac{1}{|X| - |S|} \left(\min(y) - \sum_{x_k \in S} \max(x_k) - \sum_{x_j \in X - (S \cup \{x_i\})} d_{G-\{s\}}(x_i, x_j) \right) \right\rceil \quad (3)$$

We are now in position to define an algorithm for the computation of $\underline{x_i}$ in an iterative way. The key idea of the algorithm is that we can find the value of $\underline{x_i}$ by seeking for the minimal set S consistent with Equation 3. Let T be a set of variables and

$$\alpha(T) = \left\lceil \frac{1}{|X| - |T|} \left(\min(y) - \sum_{x_k \in T} \max(x_k) - \sum_{x_j \in X - (T \cup \{x_i\})} d_{G-\{s\}}(x_i, x_j) \right) \right\rceil$$

If T is minimal and if T is the set of variables x_j for which $\alpha(T) + d_{G-\{s\}}(x_i, x_j) > \max(x_j)$ then $\underline{x_i} = \max(\min(x_i), \alpha(T))$. A initial set T can be defined by the variables x_j for which $\min(x_i) + d_{G-\{s\}}(x_i, x_j) > \max(x_j)$. Then, the following procedure is repeated: compute $\alpha(T)$ and compute the new set T according to the value of $\alpha(T)$. This procedure is repeated until a fix point is reached, that is T is no longer modified by the new procedure. Algorithm 2 implements this mechanism. The point is that new variables may be added to T when the value of $\alpha(T)$ is shifted up. Indeed, $x_i = \alpha(T)$ belongs to less and less shortest paths, as the value of x_i increases. Note that the value of $\alpha(T)$ computed by the function `Lower-bound` may be smaller than $\min(x_i)$ when $\min(y)$ does not introduce any constraint. That's why this function returns $\max(\alpha(T), \min(x_i))$.

Algorithm 2 computes $\underline{x_i}$ in an at most n iterations since at least one variable is put in T at each step. The two sums can be updated in $O(1)$ when a new element is added to T.

Algorithm 2: Computing $\underline{x_i}$

Function(`Lower-bound`(`IN`: Δ, x_i, `OUT` :$\underline{x_i}$))
% Δ : sorted list of $\Delta_j = \max(x_j) - d_{G-\{s\}}(x_i, x_j)$
$T \leftarrow \emptyset$
$\alpha(T) \leftarrow \min(x_i)$
repeat
  $T \leftarrow T \cup \{x_j : \Delta_j < \alpha(T)\}$

$$\alpha(\underline{x_i}) \leftarrow \left\lceil \frac{1}{|X| - |T|} \left(\min(y) - \sum_{x_k \in T} \max(x_k) - \sum_{x_j \in X - (T \cup \{x_i\})} d_{G-\{s\}}(x_i, x_j) \right) \right\rceil$$

until T does no more change
return $\max(\alpha(T), \min(x_i))$

Before starting this algorithm we have to compute the shortest paths $d_{G-\{s\}}(x_i, x_j)$. The point is that the $d_{G-\{s\}}(x_i, x_j)$ can be computed on the graph of reduced costs. (See Property 5.) Moreover, these shortest path distances have only to be computed once since they do not depend on the values of the domains. So, the greatest value of x_j which is consistent with $\underline{x_i}$ can be determined by computing $d_{G-\{s\}}(x_i, x_j)$ on the graph of reduced costs. No propagation step is required: when $\min(x_i)$ is increased it is useless to reconsider $\min(x_j)$ if x_j has been updated before x_i during step 3 of the interval consistency filtering algorithm. (See Algorithm 1.) This results from Proposition 4 which states that $-d(x_i, s)$ is the lower bound of $D^*(x_i)$.

Here is a short example that illustrates the process performed by algorithm 2. Consider the CSP P_1 defined by the following distance constraints :

$D^*_{(x_1)} = [3, 10]$ $D^*_{(x_2)} = [1, 3]$ $D^*_{(x_3)} = [3, 5]$ $D^*_{(x_4)} = [5, 8]$ $D^*_{(x_5)} = [8, 10]$

$x_2 \leq x_3 - 1 \quad x_3 \leq x_4 - 2 \quad x_4 \leq x_5 - 2 \quad x_1 \leq x_4 + 2$

The distance graph associated to the CSP P_1 is given by Figure 2.

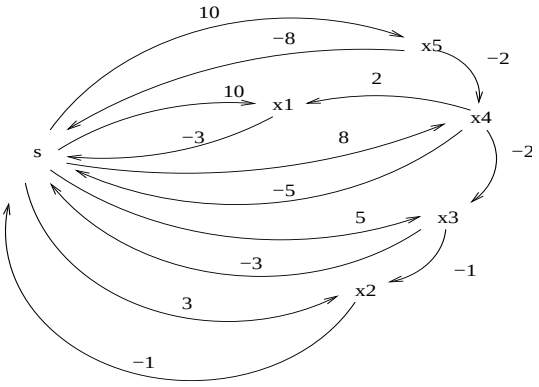


FIGURE 2. Distance graph associated to the CSP P_1

Assume that we want to compute $\underline{x_4}$ after a modification of $\min(y)$. The data compute before the algorithm starts are reported in Table 1.

i	$\max(i)$	$d_{G-\{s\}}(x_4, i)$	$\Delta_i = \max(i) - d_{G-\{s\}}(x_4, i)$
1	10	2	8
2	3	-3	6
3	5	-2	7
5	10	∞	$-\infty$

TABLE 1

Suppose that $\min(y)$ is set to 26. Since $\min(x_4) = 5$, variables x_5 is added to T. Thus, $\alpha(T) \leftarrow \lceil \frac{1}{4}(26 - (10) - (2 - 3 - 2)) \rceil = 5$. No variable is added to T and the algorithm returns $\min(x_4)$.

Now suppose that $\min(y)$ is shiffted up to 35. The set T still contains x_5 at the beginning but now $\alpha(T)$ is equal to $\lceil \frac{1}{4}(35 - (10) - (2 - 3 - 2)) \rceil = 7$. Thus, x_2 is added to T and $\alpha(T)$ is set to $\lceil \frac{1}{3}(35 - (10 + 3) - (2 - 2)) \rceil = 8$. Then, x_3 is added to T and $\alpha(T)$ is set to $\lceil \frac{1}{3}(35 - (10 + 3 + 5) - (2)) \rceil = 8$. The fixed point is reached and the algorithm returns 8 that actually corresponds to the smallest value of x_4 which is interval consistent in the constraint system $P_1 \cup \{\min(y) = 35\}$.

5.1. Complexity issues

The shortest distance between every x_i and all other nodes can be computed with Dijkstra's shortest path algorithm on the graph of reduced costs in $n \times O(m + n \log n)$ Of course, since we have to check whether the graph contains negative cycles the cost of the shortest paths between the first considered x_i and all other nodes will be in $O(nm)$. The shortest paths have only to be recomputed when the constraints changes [1].

Δ_j can be computed in $O(n)$ time. Sorting Δ costs $O(n \log n)$ whereas $\alpha(\underline{x_i})$ can be computed in constant time at each iteration step. Then, the computation of $\underline{x_i}$ can be achieved in $O(n)$ time.

So, the cost enforcing interval consistency on the IS constraint is $n \times O(m + n \log n)$ time.
Maintaining interval consistency on IS after the modification of some bound can be done in $n \log(n)$ time.

6. Discussion

It is also instructive to remark that our algorithm still works when the function to be optimized is of the form $y = \Sigma_{i=1}^{i=n} \alpha_i x_i$ where the α_i are non-negative real numbers. However, interval consistency of IS can no longer be established in polynomial time since in this case the sum constraint becomes NP-Complete:

Proposition 10. *Finding a tuple on the variables of X such that $\sum_{x_i \in X} a_i x_i = v$ is an NP-Complete problem even if the domain of the variables of X are interval of integers.*

Proof:
This problem is obviously in NP (easy polynomial certificate). We transform SUMSET-SUM to this problem. SUMSET-SUM is: *Instance*: finite set A, size $s(a_i)$ in Z+ for each $a_i \in A$, positive integer k. *Question*: is there a subset A' of A such that the sum of the sizes of the elements in A' is exactly k?

[1] Constraints may change at the branching step in optimization problems; for instance, a constraint $|x - y| \geq 5$ yields two constraint systems: one with the constraint $x - y \geq 5$, and one with the constraint $y - x \geq 5$.

For each $a_i \in A$ we define a variable x_i whose domain is the interval of integers $[0, 1]$. Then $sum_{x_i \in X}(s(a_i)x_i) = k$ exactly solves SUBSET-SUM.
⊙

Thus Corollary 1 cannot be extended for the case where the function to be optimized is of the form $y = \Sigma_{i=1}^{i=n} \alpha_i x_i$. Nevertheless, we can modify the previous algorithm in order to obtain a weaker filtering algorithm.

To capture the exact contribution of each x_i in the sum when the α_i are different from the value 1, we need only introduce the coefficient of x_i in Equation 1:

$$\underline{x_i} = \frac{1}{\alpha_i}\left(\min(y) - \sum_{x_j \in X - \{x_i\}} \alpha_j \max_{(x_i \leftarrow \underline{x_i})}(x_j)\right) \tag{4}$$

In Section 1, we defined $\mathcal{I}_{neq}$ as the subset of binary inequalities that involve only variables occurring in the objective function $f(x)$. However, $\mathcal{I}_{neq}$ could be extended to the subset of binary inequalities that involve either variables occurring in $f(x)$ or variables connected to variables occurring in $f(x)$. For instance, assume that x_1 and x_2 occur in the objective and let $\{x_1 \leq y_1 + c; y_1 \leq y_2 + c'; z_1 \leq z_2 + c''\}$ be the set of binary inequalities. Then, this extended set of inequalities would contain $\{x_1 \leq y_1 + c; y_1 \leq y_2 + c'\}$.
Let S_x be the set of variables occurring in $f(x)$. To capture this extension, we need only to replace X by S_x in $\sum_{x_j \in X - \{x_i\}}$ of Equations 1 and 5. Considering this extended set of inequalities may entail a better pruning of the domains.

7. Conclusion

This paper has introduced a new global constraint which handles as a single constraint a sum constraint and a system of binary linear inequalities. An efficient algorithm has been proposed to achieve an interval–consistent filtering of this new global constraint. The cost of this algorithm is not higher than the cost of a filtering algorithm which handles only the inequalities. A direct application of this constraint concerns optimization problems where it introduces a kind of "back" propagation process.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows.* Prentice Hall, 1993.
[2] N. Beldiceanu and E. Contejean. Introducing global constraints in chip. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
[3] J. Blazewicz, K. Ecker, G. Schmidt, and J. Weglarz. *Scheduling in Computer and Manufacturing Systems.* Springer Verlag, 1993.
[4] J. Carlier and E. Pinson. A practical use of jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26(12):269–287, 1990.

[5] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, January 1991.
[6] Moshe Dror, Wieslaw Kubiak, and Paolo Dell'Olmo. Scheduling chains to minimize mean flow time. *Information Processing Letters*, 61(6):297–301, 28 March 1997.
[7] Pascal Van Hentenryck and Yves Deville. The cardinality operator: A new logical connective for constraint logic programming. In *Proc. of ICLP'91*, pages 745–759, 1991.
[8] Pascal Van Hentenryck, Vijay Saraswat, and Yves Deville. Design, implementation, and evaluation of the constraint language cc(FD). *Journal of Logic Programming*, 37(1-3):139–164, October 1998.
[9] Michel Rueher J-C. Régin. A global constraint combining a sum constraint and difference constraints. In *Proc. of CP'2000,Sixth International Conference on Principles and Practice of Constraint Programming, LNCS 1894 (Springer Verlag)*, pages 384–395, Singapore, 2000.
[10] Charles E. Leiserson and James B. Saxe. A mixed-integer linear programming problem which is efficiently solvable. *Journal of Algorithms*, 9(1):114 – 128, 1988.
[11] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proc. of AAAI94*, pages 362–367, Seattle, Washington, 1994.
[12] J-C. Régin. Generalized arc consistency for global cardinality constraint. In *Proc. of AAAI-96*, pages 209–215, Portland, Oregon, 1996.
[13] H. Simonis. 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, 1996.
[14] Robert E. Tarjan. *Data Structures and Network Algorithms*. CBMS 44 SIAM, 1983.
[15] P. Van Hentenryck, Yves Deville, and Choh-Man Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2–3):291–321, October 1992.

Contents

Chapitre 10

Documents Administratifs

Vous trouverez dans les pages qui suivent :
- une photocopie de Pièce d'identité
- une copie du Diplôme d'Habilitation à Diriger des Recherches
- une copie du Diplôme de Doctorat

1. Nom : REGIN

2. Prénom : JEAN CHARLES PIERRE HENRI

3. Date et lieu de naissance : 11/01/1966
MULHOUSE (068)

8. Domicile : 934 CHEMIN DES AMES DU PURGATOIRE
06600 ANTIBES

4. Délivré par : LE PREFET (06)
A NICE

le 07/10/2004

5. N° 831121200894

Pour le Préfet
et par délégation
L'Adjointe au Chef de Bureau
REG-E 2077
Elizabeth BARRA

7. Signature du titulaire

CATÉGORIES DE VÉHICULES POUR LESQUELLES LE PERMIS EST VALABLE			DEPUIS LE	JUSQU'AU	RESTRICTIONS	MENTIONS	TIMBRE
A	A1	≤ 125 cc ≤ 11 kW	11/01/1984				
	A	≤ 25 kW ≤ 0,16 kW/kg	**********				
			**********				
B	B1		11/01/1984				
	B	≤ 3500 kg ≤ (7 + 8)	11/01/1984				
C	C		**********				
D	D		**********				
E	B		**********				
	C		**********				
	D		**********				

UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS

ATTESTATION DE REUSSITE AU DIPLOME

Le Chef de la scolarité atteste que

l' Habilitation à diriger des recherches

a été décernée à

Monsieur REGIN JEAN-CHARLES

né le 11 janvier 1966 à MULHOUSE (068)

au titre de l'année universitaire 2004/2005

Titre des travaux	: Modélisation et contraintes globales en programmation par contraintes
Date de soutenance	: 16 novembre 2004
Etablissement soutenance	: Université de Nice-Sophia Antipolis
Jury	: M. Michel COSNARD, Président du jury, Directeur de Recherches - INRIA M. JACQUES CARLIER, Rapporteur du jury, Professeur des Universités M. YVES CASEAU, Membre du jury, Docteur d'Etat M. Alain COLMERAUERE, Membre du jury, Professeur des Universités M. FRANCOIS FAGES, Membre du jury, Directeur de Recherches - INRIA M. Michel RUEHER, Membre du jury, Professeur des Universités Université de Nice-Sophia Antipolis
Ecole doctorale	: Sciences et Technologies de l'Information et de la Communication

Fait à Nice, le 30 novembre 2004

UNIVERSITE DE NICE-SOPHIA ANTIPOLIS 1.1.1996

MARTINEZ Stéphane

N° étudiant : 20408223

Avis important: Il ne peut être délivré qu'un seul exemplaire de cette attestation. Aucun duplicata ne sera fourni.

REPUBLIQUE FRANCAISE
Ministère de la Jeunesse,
de l'Education Nationale
et de la Recherche

ACADEMIE DE MONTPELLIER

Université MONTPELLIER II
SCIENCES ET TECHNIQUES DU LANGUEDOC

ATTESTATION

Le Président de l'Université MONTPELLIER II atteste que

Monsieur REGIN Jean-Charles
Né le 11 janvier 1966 à Mulhouse (Haut-Rhin)

En application de l'arrêté du 30 mars 1992 relatif aux études doctorales, a soutenu

devant ladite Université, le 21 décembre 1995

une thèse en vue d'obtenir le grade de

DOCTEUR de l'UNIVERSITE MONTPELLIER II

Formation Doctorale : **Informatique**

Le Jury a déclaré admis(e) Monsieur REGIN Jean-Charles

et lui a décerné la mention : « **TRES HONORABLE avec les Félicitations du Jury**»

Fait à Montpellier, le 10 décembre 2004

UNIVERSITE MONTPELLIER II ★ SCIENCES ET TECHNIQUES DU LANGUEDOC

Pour le Président
La Directrice associée
de la D.R.E.D.

M. VIANEY-LIAUD

Nota : Il ne peut être délivré qu'un seul original de la présente attestation. Aucun duplicata ne sera fourni par l'administration.