RAIRO Operations Research
RAIRO Oper. Res. **39** (2005) 87–103
DOI: 10.1051/ro:2005006

ALGORITHMES HYBRIDES GÉNÉRIQUES POUR LA RÉSOLUTION DE PROBLÈMES DE SATISFACTION DE CONTRAINTES

Hervé Deleau[1], Jin-Kao Hao[1] et Frédéric Saubion[1]

Abstract. In this paper, we present a generic hybrid algorithm for combining complete (constraint programming) and incomplete (local search) methods in order to solve constraint satisfaction problems. This algorithmic scheme uses constraint propagation techniques and local search heuristics over populations. The structures involved provide an harmonious interaction between the different methods, and also benefit from the respective methods' assets. We propose various combination strategies and emphasize their interest on some examples which are solved by means of an implementation.

Résumé. Nous présentons dans cet article un algorithme générique hybride permettant de combiner des méthodes complètes (programmation par contraintes) et incomplètes (recherche locale) pour la résolution de problèmes de satisfaction de contraintes. Ce schéma algorithmique basé sur la gestion de populations, utilise des techniques de propagation de contraintes intégrant également des heuristiques de recherche locale. Les structures utilisées autorisent une interaction homogène entre les différentes méthodes mises en œuvre et permettent également de bénéficier de leurs atouts respectifs. Nous proposons alors diverses stratégies de combinaisons dont nous mettons en avant l'intérêt sur quelques exemples par le biais d'une implémentation.

Mots Clés. Satisfaction de contraintes, propagation de contraintes, méthodes de recherche hybride.

Reçu le 9 mars 2004. Accepté le 8 février 2005.

[1] LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers, France ;
deleau@info.univ-angers.fr ; hao@info.univ-angers.fr ;
saubion@info.univ-angers.fr

1. Introduction

Les problèmes de satisfaction de contraintes [22] (Constraint Satisfaction Problems CSP) permettent de modéliser un grand nombre d'applications (planification, ordonnancement, emploi du temps ...). Ils sont définis par un ensemble de variables pouvant prendre leurs valeurs dans des domaines particuliers et un ensemble de contraintes qui sont des relations entre les variables. La résolution du problème consiste alors à trouver une combinaison de valeurs satisfaisant ces contraintes. De nombreux algorithmes et systèmes ont été développés pour résoudre les CSP et se répartissent globalement en deux types d'approches.

Les *approches complètes* sont capables d'explorer l'intégralité de l'espace de recherche pour en extraire toutes les solutions ou bien, le cas échéant, pour détecter l'absence de solution. Cette complétude est toutefois obtenue en contrepartie d'un coût calculatoire important, ce qui limite bien souvent leur utilisation dans le cadre des problèmes de grande taille ou des problèmes très contraints avec un nombre très faible de solutions.

Les *approches incomplètes* quant à elles, reposent essentiellement sur l'utilisation d'heuristiques dédiées permettant d'explorer rapidement des zones variées de l'espace de recherche afin d'y trouver éventuellement des solutions. Alors que cette démarche privilégie une efficacité calculatoire, elle ne permet en aucun cas de garantir l'obtention de solutions ni, *a fortiori*, de constater leur absence.

Un principe largement partagé pour la conception d'algorithmes toujours plus performants et robustes, consiste à tirer parti des avantages respectifs des différentes approches de résolution d'un problème et conduit donc souvent à les combiner.

L'hybridation des deux approches décrites plus haut a déjà été étudiée dans le cadre de la résolution de CSP [7,12,13,18,19,21,23]. Ces dernières années d'autres types d'hybridation ont été développés ; citons les travaux sur la combinaison de la programmation linéaire en nombre entier et la programmation par contraintes pour traiter les problèmes de planification et d'ordonnancement en production [15]. Mais, souvent, cette hybridation se révèle relativement hétérogène dans sa formulation, méthodes complètes et incomplètes utilisant des structurations différentes de l'espace de recherche. L'objectif de cet article est de proposer un cadre générique permettant d'intégrer ces divers éléments.

Résoudre un CSP consiste à parcourir un ensemble d'affectations possibles de valeurs aux variables afin d'en extraire des solutions. Nous considérerons cet ensemble comme une population d'individus soumis à diverses méthodes de résolution issues de la programmation par contraintes [10] et de la recherche locale [1]. Nous disposerons d'une population permettant de couvrir en permanence les zones contenant des solutions et qui évoluera de manière à permettre l'exploration de zones de plus en plus précises. Cette population sera utilisée pour générer des points sur lesquels nous intensifierons la recherche afin d'atteindre de possibles solutions par le biais d'un mécanisme de recherche locale. Nous proposons alors un algorithme hybride générique que nous instancions avec diverses stratégies de combinaison qui sont ensuite testées et évaluées sur un ensemble de problèmes.

2. Un algorithme hybride pour la résolution des CSP

Nous rappelons ici une formulation générale des problèmes de satisfaction de contraintes (CSP) [22]. On se donne un ensemble de variables $X = \{x_1, \cdots, x_n\}$ dont les domaines respectifs sont dans l'ensemble $D = \{D_1, \cdots, D_n\}$. Une contrainte quelconque est une relation $c \subseteq D_1 \times \cdots \times D_n$. Un CSP est donc classiquement défini par un triplet (X, D, C) où C est un ensemble de contraintes. Une affectation est une fonction $v\colon X \to \bigcup_{i=1}^{n} D_i$ telle que $\forall 1 \leq i \leq n, v(x_i) \in D_i$. Une affectation v est une solution du CSP (X, D, C) si et seulement si $\forall c \in C, (v(x_1), \cdots, v(x_n)) \in c$. Nous notons Sol l'ensemble des solutions du CSP.

Nous définissons à présent un cadre générique permettant d'intégrer des méthodes de résolution complètes et incomplètes dans un schéma algorithmique commun. En nous inspirant des méthodes évolutionnistes [11], notre algorithme opère sur des populations d'individus représentant des portions de l'espace de recherche du CSP initial. Les méthodes que nous mettons en œuvre opèrent de manière complémentaire sur cette structure. D'une part, les méthodes complètes, utilisant la propagation de contraintes et la découpe de domaines (suppression des valeurs inconsistantes du domaine de la variable), ont pour but de réduire la taille de ces zones tout en conservant l'ensemble des solutions du CSP initial. D'autre part, les méthodes de recherche locale visent à explorer les points contenus dans ces zones, de proche en proche, selon une fonction de voisinage, afin d'atteindre une éventuelle solution. Notre algorithme procède donc en plusieurs phases et des opérateurs de sélection permettent de modéliser diverses stratégies de recherche.

2.1. Représentation et espace de recherche

Pour un CSP (X, D, C), une approche classique consiste à considérer comme espace de recherche l'ensemble des valuations possibles qui peut donc être assimilé à l'ensemble des n-uplets $S = D_1 \times \cdots \times D_n$ de valeurs possibles pour les variables.

Nous utilisons, pour couvrir cet espace de recherche, une population constituée d'individus représentant des parties de l'espace de recherche, associant à chaque variable x_i, non plus une valeur, mais un sous-ensemble d_i de son domaine D_i.

Définition 1 (individus)**.** Étant donné un CSP (X, D, C) l'ensemble des individus est défini par :

$$I = \{(d_1, \cdots, d_n) \mid \forall 1 \leq i \leq n, d_i \subseteq D_i\}.$$

Nous distinguons ici des individus particuliers dont les domaines sont des singletons.

Définition 2 (point)**.** Un individu $\xi = (d_1, \cdots, d_n) \in I$ tel que $\forall 1 \leq i \leq n, |d_i| = 1$ est appelé un point.

Remarquons que nous avons alors des inclusions possibles entre les individus et que nous pouvons définir l'ensemble des points associés à un individu.

Définition 3 (ensemble de points)**.** Soit $\xi = (d_1, \cdots, d_n) \in I$, $point(\xi) = d_1 \times \cdots \times d_n$ est l'ensemble des points inclus dans ξ.

La notion de solution d'un CSP concerne ici les points et nous en déduisons immédiatement qu'un point ξ est une solution ssi $\forall c \in C, \xi \in c$. Un individu ξ sera inconsistant ssi $\exists 1 \leq i \leq n, d_i = \emptyset$.

Nous noterons $Sol(\xi) = point(\xi) \cap Sol$ l'ensemble des solutions du CSP initial qui sont dans ξ. Une population est un ensemble d'individus et $\Pi = 2^I$ est donc l'ensemble des populations. L'ensemble des solutions contenues dans une population $\pi \in \Pi$ sera $Sol(\pi) = \bigcup_{\xi \in \pi} Sol(\xi)$.

Notre objectif est à présent d'appliquer à ces populations des opérateurs que l'on pourra classer en deux groupes selon qu'ils vont s'appliquer sur des régions de l'espace ou sur des points (correspondant respectivement aux approches complètes et incomplètes). Nous présentons d'abord le schéma général et les propriétés de nos opérateurs.

2.2. Propagation de contraintes

Les méthodes complètes sont principalement basées sur une exploration systématique de l'espace de recherche. Afin d'éviter l'explosion combinatoire d'une telle exploration, ces méthodes utilisent des heuristiques spécifiques afin d'élaguer l'espace de recherche. La plus populaire de ces techniques, la propagation de contraintes, est basée sur des propriétés de consistances locales (par exemple, consistance de noeud ou consistance d'arc [14,16,17]) relatives aux contraintes, et permettant aux mécanismes de résolution de supprimer des domaines des variables les valeurs ne pouvant mener à des solutions.

Afin de conserver ce type de propriétés sur un ensemble de contraintes C, les algorithmes de recherche complète utilisent la propagation de contraintes associée au découpage. La propagation de contraintes consiste à examiner un sous-ensemble (habituellement une seule contrainte) C' de C, à supprimer les valeurs inconsistantes (du point de vue de la consistance locale) des domaines des variables apparaissant dans C' et à propager cette réduction de domaine sur les variables appartenant à $C \setminus C'$. Une fois qu'aucune propagation n'est plus possible et que les solutions ne sont pas atteintes, le CSP est découpé (généralement, le domaine d'une variable est découpé en deux sous-ensembles, formant deux sous-CSP) en sous-CSP sur lesquels est appliquée de nouveau une propagation, et ce, jusqu'à ce que les solutions soient atteintes. Ce principe est intégré dans un processus de recherche par *backtracking* qui construit un arbre de recherche en sélectionnant à progressivement des variables auxquelles des valeurs sont affectées et qui autorise l'élagage des branches inutiles.

Nous présentons maintenant les caractéristiques des fonctions utilisées pour assurer la propagation et réduire les domaines. Dans cet article, nous proposons deux types d'opérateurs basiques pour modéliser ce processus de recherche, l'un pour la propagation, l'autre pour la découpe.

2.2.1. *Réduction de domaines*

Un opérateur de réduction s'applique sur des individus et se présente donc comme une fonction $reduc : I \rightarrow I$ telle que si $\xi = (d_1, \cdots, d_n)$ alors $reduc(\xi) =$

$(d'_1, \cdots, d'_n)$ avec $\forall i, d'_i \subseteq d_i$ et $Sol(reduc(\xi)) = Sol(\xi)$. Ces propriétés assurent que les opérateurs de réduction réduisent les domaines et conservent les solutions (voir [3, 5]).

2.2.2. *Découpe de domaines*

En complément des réductions, l'exploration peut également être poursuivie en découpant cet individu en individus plus petits puis en continuant la recherche de solutions dans chacun des nouveaux individus ainsi générés. Nous spécifions ici des fonctions qui découpent un seul domaine à la fois. Une fonction de découpe sera une fonction $decoupe : I \times D \rightarrow \Pi$ telle que $decoupe(\xi, d) = \{\xi_1, \cdots, \xi_m\}$ avec $\xi = (d_1, \cdots, d, \cdots, d_n)$ et $\forall 1 \leq k \leq m, \xi_k = (d_1^k, \cdots, d^k, \cdots, d_n^k), d^k \subset d$ et $d = \bigcup_{k=1}^{m} d^k$, vérifiant également $\forall 1 \leq k \leq m, d^k \neq \emptyset$ (sinon, $decoupe(\xi, d) = \emptyset$). Nous imposons de plus que les opérations de division permettent d'atteindre tous les points (elles autorisent la découpe des domaines jusqu'à l'obtention des points).

L'application successive de ces fonctions basiques (*reduc* et *decoupe*) permet d'implémenter un solveur complet capable d'atteindre toutes les solutions ou de prouver l'insatisfiabilité du CSP. Notre but est de coupler ce mécanisme d'exploration exhaustif de l'espace de recherche à des stratégies de recherche locale, basées sur des heuristiques spécifiques d'exploitation de cet espace de recherche dans le but d'atteindre des solutions de manière plus rapide.

2.3. Recherche locale

Le but de la recherche locale est ici d'explorer les individus afin d'optimiser un critère caractérisant la notion de solution d'un CSP. Généralement, ce critère peut être formalisé par l'ensemble des contraintes non satisfaites par un point. Nous utiliserons la recherche locale pour minimiser ce critère sur des points contenus dans un individu. Notre objectif est de présenter les principaux composants d'une recherche locale sans entrer dans ses détails algorithmiques. Nous définissons donc la fonction d'évaluation suivante sur les points :

Définition 4 (évaluation)**.**

$$\begin{aligned} &eval\colon D_1 \times \cdots \times D_n \rightarrow IN \\ &p \mapsto |\{c \in C \mid p \notin c\}| \end{aligned}$$

telle que $\forall p, eval(p) = 0 \Leftrightarrow p \in Sol$. Nous devons à présent, selon le principe fondateur des méthodes de recherche locale, pouvoir nous déplacer de proche en proche parmi les points afin de rechercher des éléments toujours plus adaptés au critère précédent. Ces déplacements s'opèrent selon une fonction de voisinage $\mathcal{N} : I \rightarrow \Pi$. Une recherche locale sera donc une fonction $rl : I \rightarrow I$ telle $rl(\xi) \in point(\xi)$ qui renvoie le meilleur point obtenu par une procédure de recherche locale sur l'espace de recherche $point(\xi)$. Si ξ est inconsistant alors $rl(\xi) = \perp$ (où $\perp$ est un point non solution que nous ajoutons à I). Étant donné un individu ξ nous

pouvons alors distinguer deux stratégies élémentaires basiques à appliquer sur les points :

- l'intensification : on applique une fonction $int : point(\xi) \rightarrow point(\xi)$ telle que $int(p) = (p')$ avec $p' \in \mathcal{N}(p) \cap point(\xi)$ et $eval(p') < eval(p)$;
- la diversification : on applique une fonction $div : point(\xi) \rightarrow point(\xi)$ telle que $div(p) = (p')$ avec $p' \in \mathcal{N}(p) \cap point(\xi)$.

Une fonction rl est composée à partir de ces stratégies en utilisant un mécanisme de contrôle spécifique.

2.4. Fonctions de sélection

Afin de définir diverses stratégies opérationnelles dans notre algorithme, il est nécessaire de sélectionner les éléments sur lesquels les fonctions précédentes vont s'appliquer. Deux cas sont alors possibles : elles s'appliquent sur un individu d'une population ($reduc$ et rl) ou elles s'appliquent sur un domaine particulier d'un individu d'une population ($decoupe$). Une fonction de sélection d'un individu dans une population sera donc de la forme $select_{ind} : \Pi \rightarrow I$ et une fonction de sélection de domaine de la forme $select_{dom} : I \rightarrow D$.

2.5. Algorithme générique hybride

Nous présentons ici un algorithme générique où nous utilisons les fonctions précédentes. Nous introduisons un ensemble permettant de collecter les solutions trouvées au cours du processus global. Nous devons remarquer qu'une solution peut apparaître soit lors de la phase de réduction ou de division qui aura produit un point solution, soit lors de la phase de recherche locale.

Input : CSP (X, D, C)
Ensemble des solutions $S \leftarrow \emptyset$
Population initiale $\pi \leftarrow \{(D_1, \cdots, D_n)\}$
  Tant que non stop faire
    $\xi \leftarrow select_{ind}(\pi)$
    $\xi \leftarrow reduc(\xi)$
    $p \leftarrow rl(\xi)$
    Si $p \in Sol$ alors $S \leftarrow S \cup \{p\}$
    $\pi \leftarrow [\pi - \{\xi\}] \cup decoupe(\xi, select_{dom}(\xi))$
  FinTq

Notons que si $reduc(\xi) \in Sol$ alors rl est l'identité et $decoupe(\xi, select_{dom}(\xi)) = \emptyset$. De même, si ξ est inconsistant, $rl(\xi) = \perp$ et $decoupe(\xi, select_{dom}(\xi)) = \emptyset$. Dans ces deux cas, ξ est supprimé de la population. Le critère d'arrêt $stop$ mentionné ici peut être vu de différentes manières :

– $stop \Leftrightarrow (\pi = \emptyset)$ auquel cas on aura couvert tout l'espace de recherche et donc trouvé toutes les solutions.

– $stop \Leftrightarrow (S \neq \emptyset)$ on choisit d'arrêter la résolution lorsque la première solution est trouvée.

On peut également choisir de stopper lorsqu'un certain nombre de solutions ont été construites ou limiter le processus à un certain nombre d'itérations. Ces divers choix que nous pouvons combiner, nous permettent d'orienter la recherche entre complétude et incomplétude. On voit que la réponse fournie par l'algorithme est double. D'une part, on dispose de l'ensemble S des solutions trouvées et, d'autre part, on dispose également de la population π qui nous donne la couverture courante de l'ensemble des solutions restantes (*i.e.*, $Sol = Sol(\pi) \cup S$).

2.6. Instanciation des fonctions

Nous instancions ici les fonctions décrites précédemment afin de définir différentes stratégies d'hybridation des mécanismes fondamentaux de résolution. Les stratégies auxquelles nous nous intéressons ici se mettent en place principalement dans l'orientation de l'exploration de l'espace de recherche, soit au travers de l'ordre dans lequel les individus seront examinés, soit par le choix des domaines sur lesquels seront appliquées les découpes.

2.6.1. *Fonctions de sélection*

La programmation par contraintes offre une panoplie de méthodes de sélection de variables et de valeurs. Les méthodes proposées dans cet article peuvent être enrichies par de nouvelles techniques dans le cadre de l'algorithme hybride proposé ci-dessus.

Sélection d'un individu

Les fonctions de sélection d'un individu au sein d'une population permettent en particulier de simuler une exploration en largeur ou en profondeur des individus (par analogie avec un parcours arborescent). Nous définissons donc les deux opérateurs de sélection :

- $select_{ind}^{larg}(\pi) = \{\xi\}$ où ξ est l'individu le plus ancien de la population (dans ce cas, la population sera gérée comme une file).
- $select_{ind}^{prof}(\pi) = \{\xi\}$ où ξ est l'individu le plus récent de la population (dans ce cas, la population sera gérée comme une pile).

D'autre part, notre algorithme permet d'autres types d'exploration et nous définissons un opérateur de sélection par tournoi :

$select_{ind}^{tour}(\pi) = \{\xi\}$: pour trois individus ξ_1, ξ_2, ξ_3 pris aléatoirement dans π, on choisit deux points $p_1^i, p_2^i \in point(\xi_i)$ et on sélectionne le meilleur individu ξ_i relativement à une moyenne sur $eval(p_1^i)$ et $eval(p_2^i)$. Notons que le choix de trois individus a été effectué empiriquement suite à une série de tests. Bien évidemment, d'autres opérateurs similaires peuvent être utilisés.

Sélection d'un domaine

Nous définissons à présent des opérateurs permettant de sélectionner un domaine particulier (*i.e.*, une variable) dans un individu. C'est ce domaine qui sera ensuite découpé.

- $select_{dom}^{min}(\xi) = \{d\}$ où $d \in D$ est le plus petit domaine de D au sens de la cardinalité.
- $select_{dom}^{D/C}(\xi) = \{d\}$ où $d \in D$ est le domaine de D ayant le plus petit rapport entre sa taille et le nombre de contraintes où la variable correspondante intervient.

Nous introduisons une méthode de sélection spécifique qui met en avant l'interaction des mécanismes de recherche locale avec le processus de découpe.

- $select_{dom}^{dma}(\xi) = \{d\}$. Dans ce cas, nous lançons deux recherches locales (décrites ci-après) $dma_N^{\alpha}(\xi) = p_1$ et $dma_N^{\alpha}(\xi) = p_2$. $d \in D$ est alors le domaine correspondant à la variable occasionnant le plus grand nombre de conflits, relativement à la fonction *eval*, dans $p1$ et $p2$. Nous utilisons ici une recherche locale très simple qui sera fréquemment sollicitée. N représente le nombre total de mouvements effectués ($\frac{N}{2}$ pour chaque recherche locale), la complexité de cette sélection est donc directement liée à ce nombre.

2.6.2. *Fonctions de réduction*

Nous définissons une fonction $reduc : I \rightarrow I$ qui applique globalement les consistances de bornes et de noeuds [16] et qui, étant donné un individu ξ, génère le plus petit individu $reduc(\xi)$ vérifiant ces propriétés et contenant les mêmes solutions que ξ. Cela correspond alors à l'obtention du plus petit point fixe des opérateurs choisis (voir [3, 5]). De plus, afin d'accroître l'efficacité de la résolution, nous introduisons également la gestion d'une contrainte globale classique *alldiff* [2, 4, 20] permettant de traiter efficacement un ensemble de diséquations.

2.6.3. *Fonctions de découpe*

Nous définissons d'abord une bissection de domaine :

- $bissec((d_1, \cdots, d_i, \cdots, d_n), d_i) = \{(d_1, \cdots, d_i', \cdots, d_n), (d_1, \cdots, d_i'', \cdots, d_n)\}$ avec $d_i = d_i' \cup d_i''$ et $-1 \leq (|d_i'| - |d_i''|) \leq 1$.

Nous introduisons une découpe plus spécifique qui intègre une notion de consistance locale $decoupe_{AC}$. L'utilisation de la consistance d'arc [14, 17] a pour but de supprimer dans un domaine les valeurs ne pouvant mener à des solutions. Cette suppression peut amener à manipuler des unions d'ensembles de valeurs. Afin d'éviter ceci, nous utilisons cette consistance dans une phase de découpe. Nous avons donc un opérateur qui, à partir d'un individu et d'un domaine sélectionné génère plusieurs individus dont les domaines vérifient l'arc-consistance tout en conservant les mêmes solutions.

2.6.4. *Recherche locale*

Nous définissons deux opérateurs de recherche locale classiques :

- rt : correspond à une recherche Tabou [9] et est telle que $rt^l_N(\xi) = p$ où p est le meilleur point (du point de vue de $eval$) obtenu avec une recherche Tabou d'au plus N itérations et une liste de longueur l. Cet algorithme commence par choisir un point de départ p_0 dans $point(\xi)$ puis itère un algorithme tabou classique [9].
- dma^α_N : il s'agit d'une descente avec marche aléatoire [1] suivant une probabilité α. De la même manière $dma^\alpha_N(\xi)$ fournit le meilleur point p obtenu en au plus N itérations.

Nous pouvons à présent décrire les combinaisons que nous avons expérimentées et comparées.

3. Application

3.1. Instanciation de l'algorithme

Une instance de l'algorithme générique (Sect. 2.5) est définie par la donnée des quatre fonctions : sélection d'un individu, réduction, recherche locale, découpe avec sélection du domaine à découper. Nous étudierons plus particulièrement les combinaisons suivantes qui se répartissent en trois groupes : les combinaisons simulant une approche complète par backtracking ($BT*$), la recherche locale seule (RL) et les combinaisons hybrides ($H*$) (voir Tab. 1). Pour $HDMA$, dans un souci d'uniformité, nous utilisons dma comme fonction rl et les paramètres utilisés par la recherche locale dans la sélection $select^{dma}_{dom}$ sont identiques à ceux de la partie rl.

Dans le tableau 1, "-" signifie que ce composant n'est pas utilisé dans l'algorithme considéré (formellement il correspond à l'identité pour *reduc* et *decoupe*, à n'importe quelle sélection aléatoire pour les fonctions de sélection et une fonction rl retournant toujours $\perp$).

Notre but est maintenant de montrer l'intérêt de l'hybridation sur un échantillon de problèmes simples et variés (au niveau du nombre de variables, des contraintes et des domaines) et non de faire des tests de performances à grande échelle.

Problèmes choisis

Affectation : problème d'affectation bijective à 25 variables et 11 contraintes ; Les classiques *Carré magique*, sa variante *Hexagone Magique* et le problème des *N-reines* ; *Langford(k,n) :* problème d'arrangement de k ensembles de n nombres de 1 à n. Nous utilisons également le problème du *Social Golfer (SG)*, du *Round Robin Tournament (RRobin)* et le *All Intervals Series*. Ces problèmes sont décrits plus amplement dans la CSPLib [8]. Enfin, nous considérons également un problème de coloration d'échiquier [6], qui introduit de nombreuses contraintes disjonctives.

Tableau 1. Combinaisons étudiées.

Nom	$select_{ind}$	$reduc$	rl	$select_{dom}$	$decoupe$
BTL	$select_{ind}^{larg}$	$reduc$	$-$	$select_{dom}^{min}$	$bissec$
BTP	$select_{ind}^{prof}$	$reduc$	$-$	$select_{dom}^{min}$	$bissec$
RL	$-$	$-$	$rt_{500,000}^{10}$	$-$	$-$
HL	$select_{ind}^{larg}$	$reduc$	rt_{100}^{10}	$select_{dom}^{min}$	$bissec$
HP	$select_{ind}^{prof}$	$reduc$	rt_{100}^{10}	$select_{dom}^{min}$	$bissec$
$HDMA$	$select_{ind}^{prof}$	$reduc$	$dma_{100}^{0.04}$	$select_{dom}^{dma}$	$bissec$
HAC	$select_{ind}^{prof}$	$reduc$	rt_{100}^{10}	$select_{dom}^{min}$	$decoupe_{AC}$
HDC	$select_{ind}^{prof}$	$reduc$	rt_{100}^{10}	$select_{dom}^{D/C}$	$bissec$
$HTour$	$select_{ind}^{tour}$	$reduc$	rt_{100}^{10}	$select_{dom}^{min}$	$bissec$

3.2. Résultats expérimentaux

Les valeurs fournies dans les tableaux correspondent à la valeur d'une moyenne effectuée sur 50 essais pour l'obtention de la première solution. Nous mesurons le nombre total d'individus ainsi que le nombre total de mouvements effectués par la recherche locale et son écart type σ. Nous indiquons également le taux de succès et le temps d'exécution sur un cluster avec Linux et Alinka (5 noeuds comprenant chacun 2 CPU Pentium IV 2, 2 Ghz et 1 Go de RAM). BTL et BTP correspondent à des algorithmes de backtracking classiques (largeur et profondeur), tandis que RL est une recherche tabou. Les valeurs obtenues par BTL et BTP correspondent en fait au nombre de noeuds d'un arbre de backtracking avant de trouver une solution et, pour RL, au nombre de mouvements nécessaires à l'obtention d'une solution.

Le tableau 2 met en évidence l'efficacité de la phase de recherche locale dans un algorithme hybride pour construire une première solution ($H*$) comparé à un algorithme de backtracking traditionnel ($B*$), mais aussi, les avantages respectifs des différents opérateurs que nous avons combinés et qui dépendent de la structure du problème et de son nombre de solutions. L'écart type relatif au nombre d'individus générés par les méthodes $H*$ n'est pas significatif sur l'ensemble des jeux et n'a donc pas été mentionné pour clarifier la présentation.

Le tableau met aussi en avant l'efficacité de la phase de réduction/découpe pour guider la recherche locale (RL *vs.* $HL*$). On notera que lorsque le nombre d'individus visités est 1, cela signifie qu'il n'y a pas eu de découpe mais seulement une réduction.

TABLEAU 2. Obtention de la première solution.

Langford(2,4)					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	4			100%	0.00
BTP	4			100%	0.00
RL		634.06	1000.54	100%	0.03
HL	1.10	76.85	71.91	100%	0.0025
HP	1.30	102.85	81.06	100%	0.003
$HDMA$	1.75	274.95	213.75	100%	0.0095
HAC	1.35	141.30	153.43	100%	0.003
HDC	1.333	119.40	73.22	100%	0.003
$HTour$	1.15	88.5	69.62	100%	0.002

Affectation					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	119			100%	0.01
BTP	91			100%	0.01
RL		2072.78	1524.05	100%	0.97
HL	17.61	1358.72	1087.78	100%	0.59
HP	62.31	1774.05	833.51	100%	0.75
$HDMA$	27.50	1791.12	2524.44	100%	0.73
HAC	66.95	1900.00	722.63	100%	0.81
HDC	60.78	1681.57	871.18	100%	0.74
$HTour$	39.73	1357.93	914.69	100%	0.61

Carré magique 4					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	4010			100%	2.91
BTP	39			100%	0.01
RL		3423.3	2882.65	100%	2.05
HL	7.84	743	827.88	100%	0.26
HP	13.5	1056.06	921.82	100%	0.38
$HDMA$	27.5	1963.6	1139.19	100%	0.82
HAC	14.4	1040.7	859.98	100%	0.73
HDC	15.64	1159.22	1051.5	100%	0.61
$HTour$	13.18	1268.18	1406.05	100%	0.44

Hexagone 5					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	5122			100%	3.73
BTP	12120			100%	5.66
RL		32332.50	39352.63	8%	37.04
HL	1985.4	293299.8	218567.31	100%	101.83
HP	11399.40	444790.80	78.53	100%	118.51
$HDMA$	10025.33	569733.33	295099.23	100%	156.9
HAC	12113.2	1233435.2	433.42	100%	325.26
HDC	2446.80	91030.8	131.15	100%	21.212
$HTour$	2925.6	150734.4	130887.09	100%	43.82

80 Reines					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	-			-	>1h
BTP	-			-	>1h
RL		116.85	127.99	100%	47.80
HL	1	68.25	26.94	100%	28.71
HP	1	63.78	14.80	100%	27.09
$HDMA$	1	70.55	21.66	100%	29.6
HAC	1	63.76	27.08	100%	25.84
HDC	1	58.18	14.12	100%	24.21
$HTour$	1	66.88	21.26	100%	27.33

200 Reines					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	-			-	>1h
BTP	-			-	>1h
RL		182.6	19.3	100%	1267.5
HL	1.2	168.0	75.9	100%	1247.5
HP	1	136.3	26.6	100%	896.7
$HDMA$	1	163.2	86.7	100%	1063.3
HAC	1	142.8	21.6	100%	982.7
HDC	1	164.6	86.8	100 %	939.9
$HTour$	1.2	164.6	86.8	100%	1256.5

All Interval Series 14					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	-			-	>1h
BTP	82			100%	0.07
RL		171242.57	142699.91	75%	49.80
HL	25492.12	5098354.25	4608210.39	80%	1367.27
HP	14.50	2821.20	391.06	100%	1.01
$HDMA$	-	-	-	-	$> 1h$
HAC	47.10	9320.20	419.33	100%	3.29
HDC	11.80	2246.30	175.62	100%	0.77
$HTour$	8448.33	1566545.66	1473280.46	30%	413.98

All Interval Series 16					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	-			-	>1h
BTP	100			100%	0.11
RL		237186.80	138936.79	55%	75.0
HL	26419.66	5283885.00	2674410.78	100%	2597.12
HP	18.20	3509.40	514.20	100%	1.18
$HDMA$	-	-	-	-	$> 1h$
HAC	60.00	11898.50	1038.31	100%	4.82
HDC	15.00	2897.25	296.91	100%	1.19
$HTour$	-	-	-	-	$> 1h$

SG 12 équipes					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	-			-	>1h
BTP	-			-	>1h
RL		2090.7	2128.52	100%	173.44
HL	4.9	942.1	903.0	100%	94.26
HP	4.25	809.6	570.2	100%	83.52
$HDMA$	3.4	1238.7	1119.5	100%	105.31
HAC	4.55	858.4	563.6	100%	92.2
HDC	6.7	1301.8	978.4	100%	104.98
$HTour$	1.2	84	60.1	100%	12.94

SG 14 équipes					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	-			-	>1h
BTP	-			-	>1h
RL		1654.6	1044.23	55%	380.96
HL	28.6	5704.3	3220.8	70%	1289.66
HP	10.0	1981.5	618.5	50%	435.67
$HDMA$	2.5	964.0	597.0	<10%	229.0
HAC	43.0	8568.5	6369.5	<10%	1916.0
HDC	-	-	-	-	>1h
$HTour$	31.0	6162.0	3589.0	40%	1654.55

RRobin 8					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	-			-	>1h
BTP	-			-	>1h
RL		2770.3	2059.63	100%	35.54
HL	15.0	2929.8	1448.9	100%	36.66
HP	2.8	481.6	343.6	100%	5.98
$HDMA$	-	-	-	-	>1h
HAC	8.0	1552.6	1701	60%	19.04
HDC	4.5	782.5	712.5	40%	9.2
$HTour$	58.44	14525.6	11107.2	100%	172.61

Échiquier 10*10 3 couleurs					
Méthode	nb ind. gén.	nb mouv	σ	tx succès	temps(s)
BTL	-			-	>1h
BTP	-			-	>1h
RL		14067.0	14254.3	80%	284.46
HL	191.4	38195.4	21958.09	100%	836.9
HP	138.0	27524.0	15799.11	60%	724.13
$HDMA$	-	-	-	-	>1h
HAC	-	-	-	-	>1h
HDC	-	-	-	-	>1h
$HTour$	-	-	-	-	>1h

3.2.1. *Comparaison backtracking simple – hybridation ($BT*$ vs. $H*$)*

Il s'agit ici de comparer BTL et BTP qui correspondent à la simulation d'une exploration arborescente de l'arbre par un algorithme complet selon un parcours en largeur ou en profondeur avec les méthodes hybrides utilisant le même type de parcours (HL et HP).

Nous pouvons observer que systématiquement le nombre d'individus générés pour l'obtention d'une solution est réduit. Ceci est dû à l'apport de la composante *rl* qui permet d'atteindre plus rapidement une solution dans une zone donnée de l'espace de recherche. Nous pouvons de plus observer que pour certaines instances (ex. *Reines, Langford*) la phase de réduction est si efficace que la solution est atteinte par *rl* sur les premiers individus. Pour d'autres instances, le gain est moins significatif en fonction des différentes stratégies utilisées (Ex. *Hexagone* avec un parcours en largeur ou *Carré Magique* avec un parcours en profondeur). Il est de plus clair qu'une approche hybride permet de trouver des solutions là où le backtracking rencontre des problèmes d'espace mémoire ou de temps.

3.2.2. *Comparaison recherche locale – hybridation (RL vs. $H*$)*

Nous pouvons remarquer ici que le nombre de mouvements nécessaires à l'obtention d'une solution est généralement réduit pour l'hybridation. Notons, notamment sur le problème *Carré magique*, l'influence importante de la réduction de l'espace de recherche par découpe et propagation de contraintes sur le nombre de mouvements. Il convient par ailleurs de noter que l'hybridation permet souvent l'obtention d'une solution de manière systématique alors que ceci n'est pas garanti dans une approche RL seule (*cf.* tx succès). A ce propos, le nombre de mouvements indiqué pour RL ne prend en compte que les succès dans la moyenne, ce qui explique que parfois ce nombre est inférieur à celui de l'approche hybride.

3.2.3. *Fonctions de sélection des individus*

Nous avons utilisé 3 fonctions de sélection d'individu $select_{ind}^{larg}$, $select_{ind}^{prof}$ et $select_{ind}^{tour}$. Si généralement $select_{ind}^{prof}$ (correspondant à un parcours en profondeur d'abord) semble être un choix judicieux, elle perd de son efficacité sur certains problèmes (*Hexagone, Carré magique*) face à une sélection $select_{ind}^{larg}$. La sélection par tournoi, inspirée des algorithmes génétiques, semble alors un bon compromis entre ces deux modes de sélection.

3.2.4. *Fonctions de sélection des domaines et de découpe*

En ce qui concerne le choix du domaine à découper, nous avons testé deux fonctions en plus d'une sélection classique $select_{dom}^{min}$. Nos opérateurs spécifiques $select_{dom}^{dma}$ et $select_{dom}^{D/C}$ obtiennent de bons résultats (concernant le nombre d'individus générés). Toutefois ils induisent parfois un temps de calcul important. $select_{dom}^{dma}$ fournit de bons résultats sur *SG* et $select_{dom}^{D/C}$ sur *All Interval Series*.

Le domaine étant choisi, il faut maintenant déterminer la fonction de découpe. Nous pouvons comparer alors une classique bissection $bissec$ avec $decoupe_{AC}$ qui

élimine les valeurs non consistantes d'un domaine afin de faire une découpe multiple de l'individu. On s'aperçoit ici que les résultats sont assez similaires.

Nous avons par ailleurs, testé la combinaison en cascade de plusieurs fonctions au cours d'une même résolution. Cette première étude n'a pas permis d'améliorer les performances de l'hybridation et requiert une analyse plus fine des stratégies de coopération.

3.2.5. *Temps de calcul*

Nous avons déjà observé que l'hybridation des méthodes permet d'obtenir une réduction du nombre d'opérations utilisées par chacune des deux phases vis-à-vis de leur utilisation seule. Le gain en temps est quant à lui lié au temps d'exécution propre à chacune des différentes fonctions utilisées. Définir un rapport de temps entre une réduction et un mouvement recherche locale est difficile puisque chacun d'eux dépend du problème, du nombre et du type des contraintes et que nos algorithmes ne sont pas optimisés.

Si l'on compare l'utilisation de la recherche locale seule RL vis à vis d'une hybridation simple HP, le gain dû à l'hybridation peut être très important pour un problème dans lequel la réduction est rapidement efficace (*Reines, RRobin, All Interval Series*) ou pour lequel RL est mis en difficulté (*Carré magique*). Si le nombre de mouvements est plus faible alors que le nombre d'individus générés est relativement important (ex. *Echiquier*), l'avantage est alors à la recherche locale.

Si l'on compare à présent une résolution par BTP avec une approche hybride similaire HP le gain dépend de l'efficacité relative de la recherche locale. Par exemple, sur les *Reines* le gain des algorithmes hybrides $H*$ est très significatif.

Notons que l'efficacité relative d'une hybridation dont l'amélioration respective en nombre d'individus et nombre de mouvements est (α, β) est bien évidemment liée à l'efficacité individuelle de l'implémentation des méthodes propres à chacune des phases.

3.2.6. *Obtention de plusieurs solutions*

Nous avons mesuré le coût calculatoire nécessaire à l'obtention de plusieurs solutions distinctes sur un problème *Carré Magique-3* qui possède 8 solutions. La combinaison HP permet d'obtenir ces solutions en générant moins d'individus que BTP (voir Fig. 1). Dans ce cas, nous modifions la gestion de la population en supprimant progressivement les individus dans lesquels une solution a été trouvée.

D'autre part, cette possibilité d'obtenir des solutions différentes représente un réel avantage vis-à-vis de RL, ce qui peut être intéressant pour bon nombre de problèmes.

3.2.7. *Influence de la contrainte globale* alldiff

La mise en place d'une contrainte globale *alldiff* permet de traiter plus efficacement un ensemble de variables distinctes deux à deux. L'utilisation de cette contrainte induit une nette diminution du nombre d'individus générés dans le cas de BTL ou BTP mais également pour l'hybridation, même si le gain est légèrement

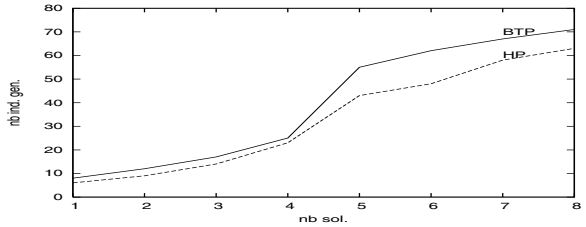

FIGURE 1. Obtention de plusieurs solutions.

moindre. Ce gain est dû au fait que cette contrainte permet de détecter plus rapidement certaines inconsistances. En revanche, l'introduction de cette contrainte globale affecte l'efficacité relative de certaines fonctions de sélection spécifiques telles que $select^{dma}_{dom}$.

L'objectif de cette expérimentation n'était donc pas de fournir des résultats compétitifs sur des problèmes complexes mais plutôt d'illustrer les atouts d'une approche hybride sur divers problèmes simples. Les hybridations $H*$ permettent d'obtenir une réduction du nombre d'opérations effectuées par chacune des deux autres approches utilisée individuellement. Le gain pourrait alors être amélioré en ajustant plus finement les paramètres agissant sur l'équilibre entre les deux phases de résolution. D'autre part, il est important de souligner qu'aucune méthode ne domine les autres sur l'ensemble des problèmes.

3.3. Extensions et particularités

Taille de la population : la taille maximale de la population permet de contrôler et de limiter la progression dans le développement des zones de recherche. Les opérateurs de sélection permettent ensuite de gérer les réductions par division. Si une taille maximale est fixée, les découpes s'arrêteront, laissant place à la recherche locale. La population permet alors par son évolution de guider la recherche locale dans la phase d'intensification.

Problème Max-CSP : d'une manière naturelle notre cadre traite le problème Max-CSP qui consiste à satisfaire le plus possible de contraintes d'un CSP donné. Il nous suffit pour cela de mémoriser lors de la phase de recherche locale la meilleure solution trouvée.

Problèmes d'optimisation sous contraintes : ce cadre général peut s'étendre aux problèmes d'optimisation sous contraintes dans lesquels on doit optimiser une fonction objective tout en satisfaisant l'ensemble des contraintes. On peut dans ce cas, utiliser la fonction objective afin de calculer une estimation de sa valeur pour les individus de la population. On peut alors en éliminer les individus dont la valeur objective estimée n'est pas intéressante. Ceci montre que les méthodes d'élagage de type *branch and bound* pourraient s'intégrer naturellement à notre schéma général.

Extension de la recherche locale : il est possible d'envisager une recherche locale sortant de la zone où elle est normalement confinée. Dans ce cas, on pourrait, par exemple, faire intervenir l'éloignement par rapport à cette zone dans la fonction d'évaluation pour contrôler cette évolution.

Algorithmique distribuée : le modèle que nous proposons ici est parfaitement adapté à une algorithmique distribuée. En effet, les populations peuvent être réparties sur plusieurs processeurs et traitées séparément.

3.4. Travaux connexes

De nombreux travaux ont été menés sur l'hybridation possibles des méthodes à base de recherche locale et des techniques de programmation par contraintes [12, 18, 19, 21, 23]. Nous retrouvons toutefois deux grandes directions dans ces travaux :

- Un mécanisme de recherche locale intervient dans un algorithme complet : à certains noeuds de l'arbre, RL est utilisée pour atteindre rapidement une solution ou améliorer une affectation et peut être, de ce fait, vu comme un mécanisme de réparation.
- Une recherche exhaustive guide RL : la propagation de contrainte est appliquée pour restreindre le voisinage ou élaguer l'espace de recherche. Les méthodes complètes peuvent également être utilisées pour explorer le voisinage et choisir le prochain mouvement.

Nous pouvons mentionner [7] qui présente un panorama général de l'utilisation possible de la recherche locale dans le cadre de la programmation par contraintes. D'autres travaux [12, 13] visent à uniformiser la combinaison de ces méthodes et leurs principaux aspects.

4. Conclusion

Nous avons présenté un algorithme générique hybride pour la résolution des CSP. Cet algorithme repose sur une collaboration entre des méthodes de propagation de contraintes et de recherche locale et sur une représentation de l'espace de recherche par des populations. Ceci permet à la fois une couverture exhaustive de l'ensemble des solutions ainsi que l'emploi de méthodes spécifiques pour intensifier la recherche des solutions.

Nous avons proposé diverses instanciations du schéma d'hybridation et montré l'intérêt d'une combinaison des approches complètes et incomplètes de résolution. Nous avons expérimenté pour cela des stratégies standard ou plus spécifiques, fondées sur des liens plus étroits entre les différentes phases de résolution. Cette étude met en avant qu'aucune combinaison ne permet d'obtenir systématiquement les meilleurs résultats sur l'ensemble des problèmes et que ces performances sont liées à la structure de ces derniers. Pourtant, l'approche hybride assure une certaine stabilité et robustesse dans le calcul des solutions.

Ce type d'approche peut être en particulier utilisé pour la mise au point de stratégies de coopération entre la programmation par contraintes et la recherche locale dans le cadre de problèmes combinatoires variés.

En outre, dans la section 3.3, nous proposons un ensemble de perspectives visant à l'amélioration de l'algorithme générique lui-même ainsi que de ses possibles applications.

Remerciements. Les auteurs remercient les relecteurs anonymes pour leurs commentaires qui ont permis d'améliorer cet article. Ce travail est en partie financé par le projet CPER COM.

Références

[1] E. Aarts and J.K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley and Sons (1997).

[2] A. Aggoun and N. Beldiceanu, Extending chip in order to solve complex scheduling problems. *J. Math. Comput. Modelling* **17** (1993) 57–73.

[3] K. Apt, From chaotic iteration to constraint propagation, in *24th International Colloquium on Automata, Languages and Programming (ICALP '97)*, Springer. *Lect. Notes Comput. Sci.* **1256** (1997) 36–55. Invited lecture.

[4] N. Beldiceanu and E. Contejean, Introducing global constraints in chip. *J. Math. Comput. Modelling* **20** (1994) 97–123.

[5] F. Benhamou, Heterogeneous constraint solving, in *Proceedings of the Fifth international conference on algebraic and logic programming, ALP'96*, Springer. *Lect. Notes Comput. Sci* **1256** (1996) 62–76.

[6] M. Clergue, J.K. Hao and F. Saubion, A chessboard coloring problem revisited, in *Proceedings of the 7th Workshop on Practical Solving of NP-complete Problems (JNPC01), Toulouse* (2001).

[7] F. Focacci, F. Laburthe, and A. Lodi, Local search and constraint programming, in *Handbook of Metaheuristics*, edited by F. Glover and G. Kochenberger, Kluwer (2002).

[8] I. Gent, T. Walsh and B. Selman, `http ://www.4c.ucc.ie/ tw/csplib/`, funded by the UK Network of Constraints.

[9] F. Glover and M. Laguna, *Tabu Search*. Kluwer Academic Publishers (1997).

[10] P. Van Hentenryck, *Constraint Satisfaction in Logic Programming*. MIT Press (1989).

[11] J.H. Holland, *Adaptation in Natural and Artificial Systems*. U. of Michigan Press (1975).

[12] N. Jussien and O. Lhomme, Local search with constraint propagation and conflict-based heuristics. *Artif. Intell.* **139** (2002) 21–45.

[13] N. Jussien and O. Lhomme, Vers une unification des algorithmes de résolution de csp, in 8^e *Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'02)* (2002) 155–168.

[14] A. Mackworth, *Encyclopedia on Artificial Intelligence*, chapter Constraint Satisfaction. John Wiley (1987).

[15] C.T. Maravelias and I.E. Grossmann, Using milp and cp for the scheduling of batch chemical processes. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Spinger. *Lect. Notes Comput. Sci.* **3011** (2004) 1–20.

[16] K. Mariott and P. Stuckey, *Programming with Constraints, An introduction*. MIT Press (1998).

[17] R. Mohr and T.C. Henderson, Arc and path consistency revisited. *Artif. Intell.* **28** (1986) 225–233.

[18] G. Pesant and M. Gendreau, A view of local search in constraint programming, in *Proc. of the Second International Conference on Principles and Practice of Constraint Programming*, edited by E. Freuder, Springer. *Lect. Notes Comput. Sci.* **1118** (1996) 353–366.
[19] S. Prestwich, A hybrid search architecture applied to hard random 3-sat and low-autocorrelation binary sequences, in *Principles and Practice of Constraint Programming - CP 2000* edited by R. Dechter, Springer. *Lect. Notes Comput. Sci.* **1894** (2000) 337–352.
[20] J.C. Régin, A filtering algorithm for constraint of difference in csps, in *National Conference of Artificial Intelligence* (1994) 362–367.
[21] P. Shaw, Using constraint programming and local search methods to solve vehicle routing problems, in *Principles and Practice of Constraint Programming - CP98*, Springer. *Lect. Notes Comput. Sci.* **1520** (1998) 417–431.
[22] E. Tsang, *Foundations of Constraint Satisfaction*. Academic Press, London (1993).
[23] M. Vasquez and A. Dupont, Filtrage par arc-consistance et recherche tabou pour l'allocation de fréquence avec polarisation, in *Actes des JNPC 2002* (2002).