Propagation de Contraintes et Listes Tabou pour le CSP

Mohammad Dib, Alexandre Caminada, Hakim Mabed

Propagation de Contraintes et Listes Tabou pour le CSP

Mohammad DIB, Alexandre CAMINADA, Hakim MABED
UTBM – SET, 90010 Belfort cedex
{mohammad.dib, alexandre.caminada, hakim.mabed}@utbm.fr

Résumé

Dans ce papier nous nous intéressons à la gestion de l'arbre de décision dans un algorithme hybride pour le CSP. Nous proposons une méthode déterministe qui permet de gérer dynamiquement des coupes dans l'arbre de recherche et l'accès aux variables de décision. Les deux principes sont basés sur la découverte et la mémorisation des associations variables/valeurs qui conduisent à des solutions infaisables. Il s'agit donc d'un système d'apprentissage qui enrichit la connaissance de l'algorithme sur le problème et augmente son efficacité au fur et à mesure de sa progression. Nous avons testé la méthode sur les benchmarks d'affectation de fréquences GRAPH et CELAR. Les premiers résultats obtenus sont au niveau des meilleurs connus actuellement sur les problèmes *Min-SPAN* et améliorent sensiblement les pourcentages de réussite.

Abstract

In this paper we focus on decision tree management for effectiveness enhancement of search within a new hybrid method. We propose a deterministic method that dynamically manages the cut of tree-branches and the access to the decision variables. Both principles are based on the discovery and the memorizing of variable/value associations that lead to unfeasible solutions. The work is thus about a learning system which progressively extends the algorithm knowledge on the problem and increases its effectiveness. We have tested the method on GRAPH and CELAR frequency assignment benchmarks. The current results are at the best-known level on the Min-Span problems but they improve the success percentages.

1 INTRODUCTION

Un problème de satisfaction de contraintes (CSP) [1] est défini par un ensemble de variables, des domaines de définition qui encadrent les valeurs que peuvent prendre les variables, et un ensemble de contraintes qui conditionnent les valeurs que pourront prendre les variables. En pratique, la famille CSP désigne un grand nombre de problèmes définis par des contraintes de ressources matérielles, temporelles, spatiales, etc. tels que les problèmes de planification, d'ordonnancement, d'affectation de ressources, etc. Les problèmes CSP ont la particularité commune d'être fortement combinatoires ; du point de vue calculatoire, les problèmes considérés induisent bien souvent des complexités algorithmiques élevées et bon nombre d'entre eux relèvent de la classe des problèmes NP-complets [2, 3]. Le travail que nous présentons se situe dans le cadre de la résolution de CSP discrets dont les variables possèdent un nombre fini de valeurs ; généralement les domaines de définition sont des sous-ensembles de *Z*. De nombreux algorithmes ont été développés pour résoudre les CSP et classiquement on identifie deux grandes familles. D'une part, les méthodes complètes qui apportent des garanties en ce qui concerne l'obtention d'une solution optimale à un problème donné, et dont souvent le temps de calcul devient rédhibitoire pour les instances de grande taille. D'autre part, les méthodes incomplètes (ou approchées) qui abordent généralement la résolution d'un CSP comme un processus d'amélioration itérative pendant lequel il s'agit de rechercher une affectation satisfaisant le plus grand nombre de contraintes. Contrairement aux approches complètes, les méthodes incomplètes ne peuvent pas conclure à la non réalisabilité d'un problème mais elles peuvent être assez rapides pour trouver une bonne solution toutefois sans garantie, ni sur la qualité des solutions proposées, ni sur le temps d'obtention de ces solutions. Lorsqu'elles sont utilisables les méthodes complètes sont donc préférables pour les garanties qu'elles apportent. Du point de vue des techniques utilisées, le panel des méthodes incomplètes s'étend des algorithmes gloutons et d'amélioration itérative les plus simples, aux méta-heuristiques les plus sophistiquées telles Recherche Tabou, algorithmes génétiques, recuit simulé…[11, 12]. L'exploration repose sur l'étude du voisinage à partir d'une solution courante. Les méthodes complètes reposent principalement sur la recherche arborescente et la notion de retour arrière (*Backtracking*). La recherche est menée par exploration d'une arborescence de décisions. A chaque itération, une solution partielle constituée d'un sous-ensemble de variables affectées est étendue progressivement en liant une variable non affectée à une valeur de son domaine. Si une des variables du problème n'a plus de valeur compatible avec la solution partielle et les contraintes présentes, on désaffecte l'une des variables précédemment affectée (*Backtrack*). Plusieurs approches ont été proposées pour améliorer les performances de l'algorithme de recherche arborescente :

- Prévenir à l'avance des affectations bloquantes par utilisation de mécanismes de propagation de contraintes [13, 14] (*LookAhead*). Nous citons en exemple les travaux de [5] sur le *Forward Checking* FC et la méthode du *Maintaining Arc-Consistency* [6].
- Repérer des variables bloquantes pour opérer les procédures de retour arrière (*LookBack*) telles que le *Conflict-Directed Backjumping* (CBJ) [7, 8] ou la méthode de *Dynamic Backtracking* DBT [9, 10].

La combinaison des deux paradigmes de résolution peut permettre de bénéficier des atouts respectifs de chacun d'entre eux pour construire des méthodes de résolution plus efficaces et plus robustes. [15] présente un survol des hybridations existantes entre les méthodes complètes et incomplètes. [16] dresse un panorama de l'utilisation de la recherche locale (LS) dans le contexte de la programmation par contrainte (PPC). [17] propose une méthode de recherche locale utilisant des mécanismes de recherche arborescente pour rendre les voisinages étendus plus intéressants. D'autres méthodes s'appuient sur la recherche locale comme la procédure de réparation des instanciations partielles [18] [30]. La méthode CN-Tabu [19] [20] utilise la Recherche Tabou combinée avec une procédure de maintien de l'arc-consistance lors du passage vers une solution voisine. Dans la majorité des algorithmes qui hybrident les méthodes complètes et incomplètes, les méthodes incomplètes sont utilisées pour ajouter une ou des décisions aléatoires. Cette décision pourra intervenir notamment au moment de définir l'ordre dans lequel les variables seront choisies pour l'affectation ou la valeur à affecter à la variable choisie. Bien souvent cet apport se traduit par une exploration plus efficace permettant de mieux éviter ou de sortir des zones de l'espace de recherche ne conduisant pas à la résolution des problèmes. En contrepartie ce type d'hybridation peut affecter les propriétés de la

méthode à reproduire la même qualité de résultat d'une exécution à une autre. Ce comportement déjà bien connu dans les méta heuristiques se retrouve ainsi dans ces méthodes hybrides. Nous avons choisi de travailler sur cet aspect en présentant une heuristique hybride qui combine des mécanismes issus de la Propagation de Contraintes avec des mécanismes issus de la Recherche Tabou. Contrairement à la plupart des autres méthodes hybrides, nous avons conçu une méthode déterministe garantissant ainsi une totale stabilité des résultats sur les problèmes d'application. La question qui nous a semblé intéressante de traiter est cependant restée celle qui fait l'apanage des méthodes hybrides à savoir l'amélioration de l'étape d'exploration pour échapper ou éviter les zones de l'espace de recherche ne conduisant pas à la résolution des problèmes. Faute d'une exploration suffisante et régulée par des mécanismes appropriés, un algorithme pourra rester longtemps, éventuellement un nombre exponentielle de fois, dans une même zone d'exploration en reproduisant des sous-ensembles d'affectation conduisant systématiquement à un échec (phénomène de *trashing*). Dans cet article nous décrivons le principe général d'une méthode hybride déterministe permettant de gérer le *trashing*, puis son fonctionnement détaillé et les premiers résultats obtenus pour l'instant sur un type de problèmes, des instances de problème d'affectation de fréquences. L'intérêt de ces instances est que l'on dispose de très nombreux résultats avec plusieurs méthodes et pour lesquelles, pour un jeu donné de ressources, le taux de succès est indiqué donc comparable.

2 LE MODELE CSP

Un problème de satisfaction de contraintes (CSP) est généralement présenté sous la forme d'un triplet *(X, D, C)* où *X* est l'ensemble des variables à affecter, *D* est l'ensemble des domaines, autrement dit pour chaque variable un ensemble de valeurs possibles, , et *C* est un ensemble de contraintes (ou de relations logiques entre variables) restreignant les valeurs que les variables peuvent prendre simultanément. La résolution d'un CSP consiste à trouver une affectation de valeur pour chaque variable de telle sorte que l'ensemble de contraintes soit satisfait (on parle de solution valide ou faisable) ; si une telle affectation existe le problème est alors dit satisfiable dans le sens où l'on peut en contenter toutes les contraintes. On peut ensuite distinguer des objectifs différents dans la résolution d'un CSP comme par exemple le fait de trouver une solution valide ou toutes les solutions valides ou de montrer qu'il n'y en a aucune. Formellement une définition de CSP est donnée par le triplet $P=(X,D,C)$ où $X=\{x_1,x_2,\ldots,x_n\}$ est l'ensemble des variables, $D=\{Dx_1,Dx_2,\ldots,Dx_n\}$ est l'ensemble des domaines, avec Dx_i qui contient l'ensemble des valeurs possibles pour x_i, et $C=\{c_1,c_2,\ldots,c_m\}$ est l'ensemble des contraintes sur les variables. Une affectation partielle est exprimée par $s = (d_{i1}, d_{i2}, \ldots, d_{ip})$ où p variables sont affectées, tel que $p = |s| <= |X|$, avec d_{ip} la valeur affectée à la variable x_{ip} de X. Une affectation est totale (ou complète) si $p = |X|$. L'espace de recherche d'un CSP $P = (X, D, C)$ que l'on notera S est l'ensemble des affectations partielles possibles. Une solution d'un CSP $P = (X, D, C)$ est une affectation $s \in S$ qui satisfait toutes les contraintes et de longueur égale à $|X|$. On note $Sol(P)$ l'ensemble des solutions de P : $Sol(P) = \{ s \in S \mid \forall c \in C, s \text{ satisfait } c \text{ et } s \text{ est complète}\}$. Une affectation partielle est dite consistante si elle ne viole aucune contrainte du problème et elle est globalement consistante si elle peut être étendue à une solution (une affectation de l'ensemble des variables ne violant aucune contrainte). Une configuration partielle est dite arc-consistante si pour chaque contrainte c_i entre les variables $(x_{i1},\ldots,x_{ir})$ il existe un ensemble de valeurs $(d_{i1},\ldots,d_{ir})$ qui satisfait la contrainte. d_{ik} doit être égale à la valeur affectée si la variable x_{ik} est déjà affectée ou doit appartenir au domaine de cette variable.

3 DESCRIPTION DE LA METHODE

3.1 PROCEDURE GLOBALE

Nous proposons ici un algorithme déterministe de propagation de contraintes qui utilise un processus de backtracking dynamique (DBT) lui-même basé sur les concepts de *Nogood* et de liste Tabou. La combinaison des concepts aura pour fonction essentielle de gérer le parcours de l'espace de recherche pour éviter ou s'échapper des phénomènes de *trashing*. Cette partie donne une description globale de la méthode dont les éléments principaux sont décrits plus précisément dans les sous-sections suivantes. La procédure présentée ci-dessous décrit le schéma général de la méthode.

```
Procedure ConstraintSatisfaction ()
1.    iter = 0;
2.    repeat
      /*Propagation*/
3.    ConstraintPropagation();
4.    if isDeadEnd()
5.     repeat
6.       Nogood =getNogood();
7.       AddNogoodInPermanentTabuList(Nogood);
8.       DesaffectVariable();
9.       AddDesaffectedVariable_
                 InTemporaryTabuList();
10.      ConstraintPropagation();
11.    until isDeadEnd()
12.   end if
13.   iter = iter+1;
      /*extend the solution*/
14.   ExtendSolution();
15.   until FindSolution() or iter >= iterMax
```

La recherche d'une solution suit un processus constructif. La recherche commence avec une solution partielle consistante ou avec une solution vide. A chaque itération, l'algorithme essaye d'étendre la solution partielle courante en conservant obligatoirement la consistance selon le principe de l'algorithme MAC [6]. L'instanciation d'une nouvelle variable implique l'ajout d'une nouvelle contrainte appelée contrainte de décision et notée xi = di. Quand une contrainte de décision devient incompatible avec l'union de l'ensemble de contraintes C et l'ensemble des contraintes de décision déjà constitué, l'algorithme réagit en réparant les décisions incompatibles par l'annulation de certaines affectations déjà réalisées. La procédure de propagation de contraintes est exécutée après chaque extension de la solution partielle courante. Cette procédure consiste à filtrer les domaines des variables non encore affectées afin de détecter à l'avance les situations de blocage ou deadend. Un deadend est détecté quand l'ensemble des valeurs possibles pour une variable donnée devient vide. Après la détection d'un deadend, l'ensemble des affectations qui a amené le domaine d'une variable à être vide est marqué. Cet ensemble d'affectations forme ce qu'on appelle un nogood [21]. Les nogood sont alors stockés dans une liste Tabou permanente qui sera utilisée pour marquer les branches bloquantes dans l'arbre de décision. Cette liste constitue la mémoire de l'algorithme des zones de l'espace de recherche qu'il a déjà visité et qui ne peuvent pas conduire à la résolution du problème. Quand un deadend est atteint l'affectation partielle doit être réparée pour être consistante. Pour ce faire une des variables présente dans le nogood est désaffectée en se basant sur des poids associés à chaque décision. Le poids d'un couple variable/valeur est le reflet des occurrences de celui-ci dans l'ensemble des nogoods archivés. Pour gérer l'exploration de l'algorithme indépendamment de l'état de la solution partielle, la valeur déjà affectée à la variable désaffectée est stockée dans une liste Tabou temporaire. La durée du statut Tabou est calculée dynamiquement pour chaque variable en fonction du nombre de fois où cette valeur a déjà été affectée à la variable. Cette

durée est utilisée afin d'éviter les cycles et constitue l'un des deux fondamentaux de la méthode de Recherche Tabou avec l'exploration complète ou partielle du voisinage, que nous n'utilisons pas du tout dans notre méthode. La méthode bénéficie ainsi de deux avantages : d'une part réduire l'espace de recherche par le filtrage et la gestion des nogoods, et d'autre part ordonner dynamiquement les variables et les valeurs à affecter pour étendre la configuration partielle en évitant les cycles via une liste Tabou de statut temporaire.

3.2 PROPAGATION DE CONTRAINTES

La propagation de contraintes consiste à réduire les domaines des variables impliquées dans les contraintes. L'objectif est de détecter et de réparer au plus tôt les solutions partielles inconsistantes. Le mécanisme de la propagation de contraintes basé sur l'arc-consistance consiste à vérifier le support pour chaque couple variable/valeur en considérant les contraintes séparément et à retirer des domaines les valeurs inconsistantes. Donnons quelques définitions :

- Arc-consistance d'un domaine : Soit P un CSP dont les domaines sont finis, et x une variable de X dont le domaine est D_x. On dit que D_x est arc-consistant ssi pour chaque contrainte c de C, toute valeur de D_x a un support dans les domaines des autres variables de c.
- Arc-consistance d'un CSP fini : On dit qu'un CSP est arc-consistant ssi tous ses domaines sont arc-consistants.

La procédure de propagation de contraintes associe pour chaque variable libre x_i un ensemble de variables affectées qui a contribué à la restriction de son domaine. L'ensemble de ces variables et leurs valeurs dans la solution partielle courante vont constituer le *nogood* à retourner si un *deadend* (domaine vide) se produit sur cette variable x_i. Le calcul du *nogood* que nous avons effectué est repris de [31]. Pour chaque variable du problème l'algorithme associe deux domaines : le domaine initial de la variable et un domaine courant qui représente le domaine dynamique de la variable après filtrage. Les domaines des variables non affectées sont réduits en prenant en considération les valeurs des variables affectées et l'incidence de la réduction des domaines des autres variables libres. Dans tous les cas l'algorithme garde la trace des origines des réductions faites sur chaque domaine. Par exemple, si la valeur d_x affectée à une variable x est en conflit avec la valeur d_y d'une variable libre y, alors d_y sera éliminée. En conséquence, la variable x est enregistrée comme une cause de la réduction du domaine de y et la cause de l'élimination de la valeur d_y. Autre exemple, si la valeur d_y d'une variable libre y n'est pas soutenue dans le domaine filtré d'une variable libre x mais est soutenu dans le domaine original de x par la valeur d_x, la valeur d_y est alors éliminée et l'algorithme enregistre que la réduction est due aux mêmes raisons que celles qui ont mené à l'élimination de la valeur d_x. Finalement, le retrait de la valeur d_y de la variable y est expliqué par l'ensemble des explications ayant mené au retrait de tous ses supports.

3.3 GESTION DES NOGOODS

Un *nogood* est l'ensemble des couples variable/valeur qui a entraîné le domaine d'une variable à être vide. Cet ensemble est donc associé à un échec de l'algorithme. Cet échec correspond à l'affectation de quelques variables correspondant à une certaine zone de l'espace de recherche dont on sait dorénavant qu'elle ne contient pas de solution au problème. Un manque d'efficacité des algorithmes est souvent du à la reproduction, parfois un nombre exponentielle de fois, des mêmes erreurs d'affectation. Lorsqu'un tel ensemble est identifié par l'algorithme il est donc important de le mémoriser pour ne pas le reproduire. Dans le but de ne pas revenir à des branches déjà testées et de ne pas tomber dans les mêmes erreurs, nous avons donc constitué une liste de *nogoods* rencontrés au cours de la recherche. La liste complète des *nogoods* contient donc tous les ensembles qui ont été détectés au fur et à mesure. La méthode de résolution apprend pendant la recherche et la liste de *nogoods* a un rôle de capitalisation de l'expérience. Nous gérons cette liste comme une liste Tabou permanente. L'algorithme prend en considération qu'une solution partielle ne doit pas contenir un sous ensemble d'affectation appartenant à la liste Tabou des *nogood*s. Selon les problèmes à traiter la liste peut devenir très importante au fur et à mesure de la progression. Nous avons donc proposé un mécanisme de réduction de la liste : à chaque nouveau *nogood*, si celui-ci domine des *nogoods* déjà stockés, les *nogoods* dominés sont éliminés ; sachant qu'un *nogood* en domine un autre s'il effectue une coupe plus sévère dans l'espace de recherche. De cette manière, l'algorithme stocke des ensembles de couples variable/valeur de différentes tailles dans une liste Tabou permanente et tous les éléments de cette liste sont exclus définitivement de la recherche.

3.4 DESAFFECTATION D'UNE VARIABLE

Lors d'un *deadend* le choix de la, ou des, variables à désaffecter est très important. Dans notre procédure, un poids est associé à chaque couple variable/valeur ; il est calculé d'une façon dynamique pendant la recherche. Après chaque détection d'un *nogood*, le poids de tous les couples variable/valeur appartenant au *nogood* est calculé selon la formule suivante, sachant que le poids initial de chaque couple est nul : $\text{poids}(x, d_x) = \text{poids}(x, d_x) + 1/|nogood|$

La cardinalité du *nogood* correspond simplement au nombre de variables qu'il contient. Après chaque *deadend* la contrainte de décision de plus grand poids dans le *nogood* courant est annulée (désaffectation de la variable). Puis pour gérer la diversité de la recherche, la contrainte de décision annulée est alors considérée interdite ou taboue pour un nombre d'itérations donnée afin d'éviter les cycles courts. La durée du statut tabou est égal au nombre de fois où la même décision a été prise par l'algorithme, et cela indépendamment des solutions partielles visitées. Nous constituons donc une liste Tabou temporaire avec une procédure déterministe qui permet à l'algorithme de diversifier sa recherche. Pour conclure cette étape, la procédure de désaffectation est immédiatement suivie d'une mise à jour du domaine de cette variable et des domaines de toutes les variables non affectées en leur remettant leur domaine initial. Puis on relance la propagation de contraintes pour mettre à jour leur domaine vis-à-vis de la solution courante. A l'issue de cette étape, tous les domaines des variables sont cohérents avec les affectations réalisées.

3.5 PROCEDURE D'EXTENSION

Il s'agit maintenant de permettre l'extension de l'affectation partielle jusqu'à l'affectation totale pour obtenir éventuellement une solution au problème. Dans notre méthode deux critères séquentiels sont employés pour guider la procédure de choix de la prochaine variable à affecter : un critère dynamique déterminé par la taille des domaines filtrés (noté MRV pour *Minimum Remaining Values*) et un critère statique se rapportant au nombre de contraintes portant sur chaque variable (degré des variables). En cas d'égalité sur leur propre domaine restant (critère MRV), le degré permet de traiter d'abord les variables a priori les plus diffusantes (critère du degré). Le choix de la valeur de la variable doit obéir à deux règles. Premièrement, la configuration partielle résultante de l'extension des affectations avec la décision $x=d_x$ ne doit pas apparaître comme un *nogood* ; ainsi le sous-ensemble de la configuration étendue ne doit pas figurer dans la liste Tabou permanente. Deuxièmement, la valeur à affecter ne doit pas être taboue à l'itération courante ; donc elle ne doit pas figurer dans la liste Tabou temporaire. Un cas particulier peut se présenter : si toutes les valeurs d'une variable choisie x figurent avec des sous-ensembles de la configuration courante dans la liste permanente des

nogoods! Dans ce cas on considère le sous-ensemble de l'affectation courante qui a entraîné la variable x à être dans ce statut comme un *nogood* à part entière. Ce sous-ensemble est l'union des explications de toutes les valeurs supprimées du domaine D_x de la variable x. Il est immédiatement ajouté à la liste Tabou permanente des *nogoods* et il est suivi de la procédure de désaffectation d'une variable pour rendre à nouveau la configuration partielle consistante (section 3.4). Enfin, si une partie des valeurs de la variable choisie x figurent dans la liste Tabou permanente avec des sous-ensembles de la solution courante, et si le reste des valeurs sont en statut Tabou temporaire du fait de l'exploration de l'algorithme, alors la variable x choisie n'est pas affectée et on fait appel à la procédure de désaffectation d'une autre variable de la configuration courante. L'exploration via la liste Tabou temporaire a donc un impact sur l'affectation partielle en cours.

4 RESULTATS

4.1 PROBLEME D'AFFECTATION DE FREQUENCES

Nous avons testé notre algorithme sur des problèmes d'affectation de fréquences (PAF). Le problème d'affectation de fréquences [22] est modélisé par un ensemble de variables représentant des émetteurs ou des liaisons radio exigeant chacun un canal de fréquence. Chaque variable est définie par un domaine de sous-ensemble discret de canaux de fréquence. Les variables du problème sont liées entre elles par des contraintes de compatibilité électromagnétique établissant les conditions de succès des communications. Ces contraintes sont souvent exprimées comme séparation minimale ou égalité exacte à respecter par l'attribution des fréquences. Les problèmes théoriques de référence sont la *k*-coloration, T-coloration, etc. Les instances que nous avons choisies pour travailler sont celles du projet européen CALMA (Combinatory ALgorithms for Military Applications). Plusieurs équipes concurrentes de chercheurs ont travaillé sur les mêmes instances de problème : 11 instances SCEN fournies par le CELAR (Centre ELectronique de l'Armement en France) et 14 autres fournies par l'université de technologie de Delft [23]. La taille des problèmes varie de 200 à 1000 variables avec des domaines de 44 valeurs et un nombre de contraintes de 1134 à 5548. En plus de la satisfaction de contraintes, quelques problèmes sont décrits avec une fonction à optimiser : le mode *Min-Span* qui consiste à minimiser la largeur du spectre utilisé, i.e. la distance entre les fréquences maximale et minimale de la solution, et le mode *Min-Order* qui consiste à minimiser le nombre de fréquences utilisées dans la solution. L'optimisation spectrale est effectuée par une relance itérative de la méthode. Pour *Min-Span*, l'algorithme commence par les domaines originaux. Si une solution faisable est trouvée, la fréquence maximale d_{max} utilisée dans la solution est supprimée des domaines et toutes les décisions de la forme $x=d_{max}$ sont annulées. La solution partielle produite ainsi est employée comme départ pour la recherche d'une nouvelle solution faisable. L'algorithme s'arrête quand la recherche d'une solution faisable a échoué pour un temps donné. La dernière solution trouvée (solution faisable de *span* minimal) est fournie comme résultat final. Pour l'optimisation *Min-Order*, le même processus de recherche est suivi mais on élimine la fréquence la moins utilisée dans la solution pour la phase suivante.

4.2 RESULTATS ET COMPARAISONS

Un nombre important de méthodes a été proposé dans la littérature pour résoudre le PAF. Nous présentons ici les méthodes recensées et comparées sur la page Web du site de l'institut de recherche Zuse Institute Berlin ZIB dédiée à CALMA (http://fap.zib.de/problems/CALMA). Quatre instances des deux jeux de test CELAR [24] et GRAPH [23] doivent satisfaire l'objectif *Min-Span* sur lequel nous travaillons pour évaluer notre nouvelle méthode hybride. Nous donnons dans le tableau 1 les résultats des 4 meilleures méthodes, parmi 8 méthodes rapportées, ainsi que nos propres résultats : en [25] un algorithme de *Branch-and-Cut* avec un modèle de programmation linéaire avec ; en [26] un algorithme de recherche Tabou ; en [27] un algorithme qui utilise une méthode de réduction de potentiel ; et pour finir, en [28] une combinaison entre une méthode de programmation quadratique suivie d'une heuristique d'arrondissement afin d'obtenir la solution finale. Les colonnes du Tableau 1 indiquent : le nom de chaque instance, son optimum connu (la fréquence maximale utilisée dans le spectre), le nombre de variables puis de contraintes, et la fréquence maximale trouvée par chaque méthode. A noter que le pas entre 2 fréquences consécutives est toujours 14. Les huit méthodes présentées sur la page Web du ZIB trouvent une solution optimale avec une fréquence maximale de 792 pour le SCEN05. Mais seulement 4 de ces 8 méthodes présentées, résolvent les instances GRAPH03, GRAPH04 et GRAPH10 et seulement 2 sur 8 résolvent chaque instance à l'optimalité. La colonne 9 (DCM) indique nos résultats ; les 4 instances sont résolues d'une façon optimale en moins de 15 secondes. Nos expérimentations ont été effectuées sur un Intel PentiumIV, 2.4GHZ, 512MO de RAM. Nous avons peu d'informations sur le contexte de résolution des méthodes au sein du projet CALMA : on ne sait pas sur quelles machines les tests ont été effectuées, ni le temps de calcul de chaque méthode, et on ne connaît pas non plus le taux de succès sur n exécutions. On retiendra donc seulement que notre méthode est au niveau des 2 meilleures sur les 8 connues en terme de résolution avec *Min-Span*.

Tableau 1. Comparaison avec plusieurs méthodes.

Scenario	Opt	V	C	[25]	[26]	[27]	[28]	DCM
SC05	792	400	2598	792	792	792	792	792
GR03	380	200	1134	380	380	-	380	380
GR04	394	400	2244	394	394	-	394	394
GR10	394	680	3907	394	394	394	397	394

Pour avoir une évaluation plus précise en taux de succès et temps de calcul, le tableau 2 montre une comparaison entre les résultats de notre méthode avec des résultats publiés en 2005 avec CN-Tabu [19][20] qui est une des meilleures méthodes référencées actuellement [29].

Tableau 2. Comparaison avec les résultats de la méthode CN-Tabu.

	V	C	[20]	T	SR	DCM	T	SR	NG
SC01	916	5548	680	1	-	680	6	20	145
SC02	200	1235	394	1	-	394	1	20	153
SC03	400	2760	666	1	-	**652**	7	20	182
SC05	400	2598	792	1	20	792	1	20	5
GR01	200	1134	408	1	-	408	13	20	1787
GR02	400	2245	394	1	-	394	7	20	1179
GR03	200	1134	380	1	20	380	5	20	50
GR04	400	2244	394	1	8	394	7	**20**	66
GR08	680	3757	**652**	15	-	680	11	20	1473
GR09	916	5246	**666**	664	-	694	35	20	776
GR10	680	3907	394	17	5	394	15	**20**	126
GR14	916	4638	352	30	-	352	**22**	20	1076

La méthode CN-Tabu a été appliquée sur les instances du tableau 2 pour *Min-Span* sur une machine plus puissante (AMD 64 3000 et 1Go de RAM équivalent à un PentiumIV 3GHz). Ce tableau donne d'abord le nom de l'instance, le nombre de variables et de contraintes. Puis les colonnes 4 [20], 5 et 6 sont pour CN-Tabu et 7 [DCM], 8 et 9 pour notre méthode. Les colonnes [20] et [DCM] donnent le meilleur résultat (la fréquence maximale utilisée dans le spectre), les colonnes T donnent le temps d'exécution en secondes, et les colonnes SR (*Success Rate*) donnent le nombre de fois où la solution optimale est atteinte sur 20 relances (par exemple 8/20 pour

GR04 avec CN-Tabu). Parfois le taux de succès de CN-Tabu n'est pas publié pour certaines instances, dans ce cas nous indiquons «-». La colonne NG indique le nombre de *nogoods* stockés dans la liste par notre méthode. On voit qu'il n'y a pas de relation directe entre la taille du problème et la taille de cette liste à la fin de l'exécution ; le problème SC01 est un des plus gros en nombre de variables et de contraintes, et pourtant la liste de nogood à l'issue de la résolution contient seulement 145 ensembles de couples variable/valeur. Les résultats présentés dans le tableau 2 montrent la compétitivité de notre méthode hybride. Lorsque les résultats sont en faveur d'une méthode la cellule est grisée. CN-Tabu est ainsi plus performante sur GR08 et GR09 en utilisant 2 fréquences de moins. La méthode DCM que nous avons mise au point est plus performante à 4 reprises : sur SC03 en utilisant 1 fréquence de moins, sur GR04 et GR10 en ayant une réussite de 100% sur les 20 exécutions et sur GR14 où elle est plus rapide alors que notre machine de test est a priori moins puissante. Globalement, cette nouvelle méthode s'avère compétitive. Dans l'état actuel de nos travaux son apport principal réside dans son comportement déterministe grâce auquel une seule exécution suffit à l'obtention de la meilleure solution trouvée par la méthode.

5 CONCLUSION

Dans ce papier, nous avons présenté une méthode hybride prometteuse qui tire ses avantages, d'une part, de la rigueur et du caractère optimal de l'approche exacte, et d'autre part, de la grande flexibilité de l'approche heuristique. Le principe de propagation nous permet de réduire l'espace de recherche et d'identifier au plus tôt les branches bloquantes. Puis nous avons défini une liste Tabou permanente qui mémorise les *nogoods* identifiés au fur et à mesure de l'exploration de l'arbre de recherche. Cette liste est exploitée comme un mécanisme d'apprentissage par l'algorithme pour ne pas revenir sur des affectations partielles non concluantes déjà visitées et éviter ainsi les effets de *trashing*. L'algorithme utilise la liste permanente pour filtrer des zones de l'espace de recherche. Il s'en suit une meilleure exploration de l'espace des solutions consistantes. Avec cette liste les variables peuvent être pesées et ordonnées selon leurs occurrences dans les *nogoods* identifiés. Ces pondérations donnent les informations nécessaire à l'algorithme sur comment et où revenir dans l'arbre de recherche. Une deuxième liste Tabou est utilisée dans l'algorithme pour éviter les cycles courts dans la recherche et les répétitions de décisions. Cette liste est temporaire et elle permet d'apporter de la diversité dans le choix des variables ou des valeurs. Nous avons testé et comparé la méthode sur les problèmes CALMA d'affectation de fréquences. Le résultat des comparaisons a montré que ce nouvel algorithme hybride est une méthode prometteuse qui tire profit de son comportement déterministe. L'efficacité de la méthode dépend de la pertinence des procédures d'extension des affectations partielles, de la propagation de contraintes et de la procédure de désaffectation basée sur les *nogoods*. L'amélioration de ces mécanismes devrait encore permettre d'améliorer les performances de la méthode. Prochainement, nous étudierons l'impact des différentes composantes de la méthode vis-à-vis des résultats et l'application de la méthode sur des problèmes différents.

REFERENCES

[1] E. P. K Tsang. Foundations of Constraint Satisfaction. Academic Press, London, 1993.

[2] M. R. Garey and D. S. Johnson. Computers and Intractability , A Guide to the Theory of NP-Completeness. W.H. Freeman & Company, San Francisco, 1978.

[3] C. H. Papadimitriou. Computational Complexity. Addison-Wesley Publishing Company, 1994.

[4] F. Bacchus and P. van Run. Dynamic variable reordering in CSPs. In Principles and Practice of Constraint Programming (CP95), number 976 in LNCS, pages 258–275, 1995.

[5] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for CSP. Artificial Intelligence, 14 :263–313, 1980.

[6] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Proc. of European Conference on Artificial Intelligence (ECAI'94), Amsterdam, pages 125–129, 1994.

[7] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. Computational Intelligence, 9(3) :268–299, August 1993.

[8] X. Chen and P. van Beek. Conflict-Directed Backjumping Revisited. Journal of Artificial Intelligence Research, 14 :53–81, 2001.

[9] M. Ginsberg. Dynamic backtracking. Journal of Artificial Intelligence Research, 1:25–46, 1993.

[10] N. Jussien, R. Debruyne and P. Boizumault. Maintaining Arc-Consistency within Dynamic Backtracking, Principles and Practice of Constraint Programming (CP 2000), Lecture Notes in Computer Science, no. 1894, pp. 249-261, Springer-Verlag, 2000.

[11] I. Devarenne, H. Mabed and A. Caminada. Analysis of Adaptive Local Search for the Graph Coloring Problem. 6th MIC, Vienna, 2005.

[12] S. Ben Jamaa, Z. Altman, J.M. Picard, and B. Fourestié. Optimisation de réseaux mobiles UMTS à l'aide des algorithmes génétiques. In J. Dréo et al. editors, Méta heuristiques pour l'optimisation difficile. Eyrolles, 2003.

[13] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In Proceedings of the International Joint Conference on Artificial Intelligence, pp 412–417, Japan, 1997.

[14] R. Debruyne. Consistances locales pour les problèmes de satisfaction de contraintes de grande taille. PhD thesis, Université Montpellier II, décembre 1998.

[15] T. Lambert. Hybridation de méthodes complètes et incomplètes pour la résolution de CSP, PhD thesis, Université de Nantes, octobre 2006.

[16] F.Focacci, F.Laburthe, and A.Lodi. Local search and constraint programming. In Fred Glover and Gary Kochenberger, editors, Handbook of Metaheuristics. Kluwe, 2002.

[17] P.Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.F.Puget, editors, Principles and Practice of Constraint Programming – CP98, 4th International Conference, number 1520 in LNCS, page 417-431. Springer, 1998.

[18] N.Jussien and O.Lhomme. Local search with CP and conflict-based heuristics. Artificial Intelligence, 139(1) :21-45, 2002.

[19] M. Vasquez, Résolution en variables 0-1 de problèmes combinatoires de grande taille par la méthode tabou. PhD thesis, université d'Angers, U.F.R. de Sciences, décembre 2000.

[20] A. Dupont. Étude d'une méta-heuristique hybride pour l'affectation de fréquences dans les réseaux tactiques évolutifs. PhD thesis, Université Montpellier II, Octobre 2005.

[21] T. Schiex, G. Verfaillie, Nogood recording for static and dynamic constraint satisfaction problems. International Journal on Artificial Intelligence Tools (IJAIT) (1994).

[22] H. Mabed, A. Caminada, J.K. Hao and D. Renaud, A dynamic traffic model for frequency assignment, LNCS, Vol.2439, pp. 779-788, 2002.

[23] H. P. van Benthem. GRAPH Generating Radio Link Frequency Assignment Problems Heuristically. PhD thesis, Delft University of Technology, 1995.

[24] http://www.inra.fr/internet/Departements/MIA/T//schiex/Doc/24.shtml

[25] K. I. Aardal, A. Hipolito, C. P. M. van Hoesel, and B. Jansen. A Branch-and-Cut Algorithm for the Frequency Assignment Problem. Research Memorandum 96/011, Maastricht University, 1996.

[26] S. R. Tiourine, C. A. J. Hurkens, and J. K. Lenstra. Local Search Algorithms for the Radio Link Frequency Assignment Problem. Telecommunication Systems, 13 :293–314, 2000.

[27] J. P. Warners, T. Terlaky, C. Roos, and B. Jansen. A Potential Reduction Approach to the Frequency Assignment Problem. Discrete Applied Mathematics, 78 :251–282, 1997.

[28] D. V. Pasechnik. An Interior Point Approximation Algorithm for a Class of Combinatorial Optimization Problems : Implementation and Enhancements. Technical report, Delft University of Technology, 1998.

[29] http://uma.ensta.fr/conf/roadef-2001-challenge/

[30] M.Dib, H.Mabed, A.Caminada and J.Hu. Propagation de Contraintes et gestion de *nogood* pour le CSP. ROADEF, 2008.

[31] N. Jussien. Programmation par contraintes avec explications. 7ièmes Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'01), pp. 147-158, 2001.