Recherche coopérative et Nogood Recording

Cyril Terrioux

LIM - Equipe INCA
39, rue Joliot-Curie
13453 Marseille Cedex 13

terrioux@lim.univ-mrs.fr

RÉSUMÉ. *Dans le cadre du problème de satisfaction de contraintes, nous proposons un nouveau schéma de recherche parallèle coopérative. La coopération est réalisée en échangeant des nogoods (instanciations qui ne peuvent être étendues en une solution). Nous associons un processus à chaque solveur et nous introduisons un gestionnaire de nogoods, afin de réguler les échanges de nogoods. Chaque solveur emploie l'algorithme Forward-Checking avec Nogood Recording. Nous ajoutons à l'algorithme une phase d'interprétation qui limite la taille de l'arbre de recherche suivant les nogoods reçus. Les solveurs diffèrent par l'emploi d'heuristiques distinctes pour le choix des variables et/ou des valeurs. L'intérêt de cette approche est démontré expérimentalement. En particulier, nous obtenons des accélérations linéaires ou superlinéaires pour des problèmes consistants comme pour des problèmes inconsistants, jusqu'à une dizaine de solveurs.*

ABSTRACT. *Within the framework of constraint satisfaction problem, we propose a new scheme of cooperative parallel search. The cooperation is realized by exchanging nogoods (instantiations which can't be extended to a solution). We associate a process with each solver and we introduce a manager of nogoods, in order to regulate exchanges of nogoods. Each solver runs the algorithm Forward-Checking with Nogood Recording. We add to algorithm an interpretation's phase, which limits the size of search tree according to received nogoods. Solvers differ from each other in ordering variables and/or values by using different heuristics. The interest of our approach is shown experimentally. In particular, we obtain linear or superlinear speed-up for consistent problems, like for inconsistent ones, up to about ten solvers.*

MOTS-CLÉS : *problème de satisfaction de contraintes, recherche parallèle, coopération, nogood.*

KEYWORDS: *constraint satisfaction problem, parallel search, cooperation, nogood.*

1. Introduction

Dans le cadre du problème de satisfaction de contraintes, une des principales tâches consiste à déterminer s'il existe une solution (affectation de toutes les variables qui satisfait toutes les contraintes). Cette tâche est un problème NP-complet. Afin d'accélérer la résolution, on peut employer des recherches parallèles. La plus simple est la *recherche parallèle concurrente*, qui consiste à lancer plusieurs solveurs (chacun utilisant des heuristiques différentes) sur le même problème au lieu d'un seul. Avec une telle méthode, on espère qu'au moins un des solveurs possède une heuristique bien adaptée au problème à résoudre. Expérimentée sur le problème de coloration de graphes par Hogg et Williams ([HOG 94]), cette approche apporte un gain par rapport à une résolution classique avec un solveur, mais ce gain semble limité. Les auteurs préconisent alors l'emploi d'une recherche parallèle avec coopération.
La *recherche parallèle avec coopération* reprend les mêmes bases que la recherche concurrente, en y ajoutant un échange d'informations entre les solveurs. L'objectif est que les informations échangées permettent de mieux guider les solveurs vers une solution, et ainsi de résoudre plus rapidement le problème. Les résultats expérimentaux sur des problèmes cryptarithmétiques ([CLE 91], [HOG 93a]) et sur le problème de coloration de graphes ([HOG 93a], [HOG 93b]) mettent en évidence un gain en temps significatif par rapport à une recherche parallèle concurrente. Dans les deux cas, les informations échangées correspondent à des débuts de solutions potentielles.

Dans [MAR 96], une coopération basée sur un échange de nogoods (affectations qui ne peuvent être étendues en une solution) est proposée. Les nogoods échangés peuvent permettre aux solveurs d'élaguer leur propre arbre de recherche. Ainsi, on peut s'attendre à obtenir plus rapidement une solution. Tous les solveurs emploient l'algorithme Forward Checking avec Nogood Recording (noté FC-NR [SCH 93]). L'implémentation réalisée réunit tous les solveurs dans un seul processus, qui simule le parallélisme. L'échange de nogoods est alors effectué grâce à une structure de données commune à tous les solveurs. Le schéma proposé par Martinez et Verfaillie est orienté vers un système monoprocesseur. Testée sur des problèmes aléatoires, la recherche coopérative apparaît comme meilleure que la recherche concurrente. Toutefois, le faible gain par rapport à un seul solveur laisse un doute sur l'efficacité d'une telle méthode, en particulier dans le cas d'une implémentation multi-processus.

Reprenant l'idée de Martinez et Verfaillie, nous définissons un nouveau schéma de coopération avec échange de nogoods, qui se veut orienté vers des systèmes dotés d'un ou plusieurs processeurs. Nous associons un processus à chaque solveur. Chaque solveur utilise FC-NR. Afin d'éviter les problèmes dus aux coûts des communications, nous introduisons un processus supplémentaire : le gestionnaire de nogoods. Son rôle est de communiquer les nogoods découverts à une partie des solveurs. Nous ajoutons à FC-NR une phase d'interprétation, qui limite la taille de l'arbre de recherche, en stoppant le développement de branches vouées à l'échec suivant les nogoods reçus.
Notre second et principal objectif est de répondre à la question restée ouverte dans [MAR 96] quant à l'efficacité d'une recherche parallèle coopérative avec échange de nogoods. Nous montrons expérimentalement l'efficacité de notre schéma.

Dans la section 2, nous rappelons les notions de base sur les CSPs, sur les nogoods et l'algorithme FC-NR. Puis, dans la section 3, nous présentons notre schéma en décrivant le gestionnaire et la phase d'interprétation. Enfin, après avoir consacré la section 4 aux expérimentations, nous donnerons nos conclusions dans la section 5.

2. Définitions et rappels

2.1. *Les principales définitions sur les CSPs*

Un *problème de satisfaction de contraintes* (CSP) est défini par un quadruplet (X,D,C,R). X est un ensemble $\{x_1, \ldots ,x_n\}$ de n variables. Chaque variable x_i prend ses valeurs dans le domaine D_i, issu de D. Ces variables sont soumises aux contraintes de C. Chaque contrainte c porte sur un ensemble $X_c = \{x_{c_1}, \ldots ,x_{c_k}\}$ de variables. A chaque contrainte c est associée une relation R_c (issue de R) qui représente l'ensemble des k-uplets autorisés sur $D_{c_1} \times \cdots \times D_{c_k}$.
Un CSP est dit *binaire* si chacune de ses contraintes n'implique que deux variables. Soient x_i et x_j deux variables, on notera c_{ij} la contrainte correspondante.
Par la suite, nous ne considérerons que des CSPs binaires. Toutefois, les idées exposées peuvent s'étendre aux CSPs n-aires.

Etant donné $Y \subseteq X$ avec $Y = \{x_1, \ldots ,x_k\}$, une *instanciation* des variables de Y est un k-uplet $(v_1, \ldots ,v_k)$ de $D_1 \times \cdots \times D_k$. Elle est dite *consistante* si $\forall c \in C, X_c \subseteq Y, (v_1, \ldots ,v_k)[X_c] \in R_c$. Elle est dite *inconsistante* sinon. Autrement dit, une instanciation est consistante si elle vérifie toute contrainte c dont toutes les variables de X_c sont dans Y. On emploie indifféremment les termes d'*affectation* et d'instanciation. On note l'instanciation $(v_1, \ldots ,v_k)$ sous la forme plus explicite $(x_1 \leftarrow v_1, \ldots ,x_k \leftarrow v_k)$. Une *solution* est une instanciation consistante de toutes les variables. Etant donnée une instance $\mathcal{P} = (X,D,C,R)$, déterminer si $\mathcal{P}$ possède une solution est un problème NP-complet.

Etant donnés un CSP $\mathcal{P} = (X,D,C,R)$ et une affectation $\mathcal{A}_i = \{x_1 \leftarrow v_1, x_2 \leftarrow v_2, \ldots ,x_i \leftarrow v_i\}$, $\mathcal{P}(\mathcal{A}_i) = (X,D(\mathcal{A}_i),C,R(\mathcal{A}_i))$ est le *CSP induit* par $\mathcal{A}_i$ sur $\mathcal{P}$ avec un filtre de type Forward Checking avec :

- $\forall j, 1 \leq j \leq i, D_j(\mathcal{A}_i) = \{v_j\}$
- $\forall j, i < j \leq n, D_j(\mathcal{A}_i) = \{v_j \in D_j | \forall c_{kj} \in C, 1 \leq k \leq i, (v_k,v_j) \in R_{c_{kj}}\}$
- $\forall j,j', R_{c_{jj'}}(\mathcal{A}_i) = R_{c_{jj'}} \cap (D_j(\mathcal{A}_i) \times D_{j'}(\mathcal{A}_i))$.

$\mathcal{A}_i$ est dite *FC-consistante* si $\forall j$, $D_j(\mathcal{A}_i) \neq \emptyset$.

2.2. *Les nogoods : définitions et propriétés*

Dans cette partie, nous rappelons les principales définitions et propriétés concernant les nogoods et l'algorithme Forward Checking avec Nogood Recording.
Un nogood correspond à une affectation qui ne peut être étendue en une solution. Plus

formellement ([SCH 93]), étant donnés une affectation $\mathcal{A}$ et un sous-ensemble J de contraintes ($J \subseteq C$), $(\mathcal{A},J)$ est un *nogood* si le CSP (X,D,J,R) ne possède pas de solution contenant $\mathcal{A}$. J est appelé la *justification* du nogood (on notera X_J les variables soumises aux contraintes de J). On désigne par *arité* du nogood $(\mathcal{A},J)$ le nombre de variables affectées dans $\mathcal{A}$. Par exemple, toute affectation inconsistante correspond à un nogood, mais la réciproque est fausse.
On note $\mathcal{A}[Y]$ la restriction de $\mathcal{A}$ aux variables communes à $X_\mathcal{A}$ et à Y.

Comme FC-NR est basé sur Forward-Checking, une inconsistance est détectée dès qu'un domaine devient vide. Aussi, les causes de l'inconsistance peuvent être multiples. Afin de calculer les justifications des nogoods, nous reprenons la notion de « value-killer » (introduite dans [SCH 93]) et nous l'étendons afin de pouvoir l'utiliser dans notre solveur multiple. Etant donnés une affectation $\mathcal{A}_i$, le CSP $\mathcal{P}(\mathcal{A}_i)$ induit par $\mathcal{A}_i$, et l'ensemble N des nogoods trouvés par d'autres solveurs, une contrainte c_{kj} ($j > i \geq k$) est une *value-killer* de la valeur v_j de D_j pour $\mathcal{A}_i$ si une des trois conditions suivantes est vérifiée :

1. c_{kj} est une value-killer de v_j pour $\mathcal{A}_{i-1}$
2. $k = i$ et $(v_k,v_j) \notin R_{c_{kj}}(\mathcal{A}_i)$ et $v_j \in D_j(\mathcal{A}_{i-1})$
3. $\{x_k \leftarrow v_k, x_j \leftarrow v_j\} \in N$

Notons que dans le cas où un seul solveur est utilisé (par exemple en séquentiel), $N = \emptyset$. On retrouve alors la définition présentée dans [SCH 93].
Supposons qu'une inconsistance soit détectée à cause d'un domaine D_i devenu vide. Dans un tel cas, les raisons de l'inconsistance (ie les justifications) correspondent à la réunion des value-killers de D_i. Le théorème suivant formalise la création des nogoods à partir des inconsistances.

Théorème 1 *Soient $\mathcal{A}$ une affectation et x_i une variable non affectée. Soit K l'ensemble des value-killers de D_i.*
S'il n'existe plus de valeurs dans $D_i(\mathcal{A})$, alors $(\mathcal{A}[X_K],K)$ est un nogood.

Ce théorème permet donc de créer des nogoods à partir des inconsistances détectées. Quant aux deux théorèmes suivants, ils rendent possible la création de nouveaux nogoods à partir de nogoods existants. Le premier théorème construit un nouveau nogood à partir d'un nogood existant.

Théorème 2 ([SCH 93]) *Si $(\mathcal{A},J)$ est un nogood, alors $(\mathcal{A}[X_J],J)$ est un nogood.*

Autrement dit, on ne conserve de l'affectation que les variables mises en cause dans l'inconsistance. Ainsi, le nogood généré a une arité limitée à son strict minimum.
Le théorème 3 construit un nouveau nogood à partir d'un ensemble de nogoods.

Théorème 3 *Soient $\mathcal{A}$ une affectation et x_i une variable non affectée. Soit K l'ensemble des value-killers de D_i. Soit $\mathcal{A}_j$ l'extension de $\mathcal{A}$ par l'affectation de x_i à la valeur v_j ($\mathcal{A}_j = \mathcal{A} \cup \{x_i \leftarrow v_j\}$).*

Si $(\mathcal{A}_1,J_1)$, ..., $(\mathcal{A}_d,J_d)$ sont des nogoods, alors $(\mathcal{A},K \cup \bigcup_{j=1}^{d} J_j)$ est un nogood.

Un nogood peut être utilisé soit pour backjumper, soit pour ajouter une nouvelle contrainte ou pour renforcer une contrainte existante. Dans les deux cas, il résulte de l'utilisation des nogoods un élagage de l'arbre de recherche.

FC-NR ([SCH 93]) explore l'arbre de recherche comme Forward Checking. Lors du parcours, FC-NR profite des échecs rencontrés pour créer et mémoriser des nogoods. Ces nogoods sont alors utilisés comme décrit ci-dessus pour élaguer l'arbre de recherche. L'inconvénient de FC-NR est que le nombre de nogoods peut être exponentiel. Nous limiterons donc le nombre de nogoods en suivant la proposition de Schiex et Verfaillie ([SCH 93]) qui consiste à ne conserver que les nogoods d'arité au plus 2.

3. Description du solveur multiple

Notre solveur multiple se compose de p solveurs séquentiels basés sur FC-NR. Ces p solveurs sont lancés en concurrence sur le même CSP. Chacun d'eux possède une heuristique différente pour l'ordonnancement des variables et/ou des valeurs. Ainsi, chacun parcourt un arbre de recherche distinct. La coopération consiste à échanger des nogoods. Un solveur peut employer un nogood produit par un autre solveur afin d'élaguer une partie de son arbre de recherche, ce qui devrait accélérer la résolution.

Lors de la recherche, les solveurs produisent donc des nogoods qui sont communiqués aux autres solveurs. Par conséquent, quand un solveur découvre un nogood, il doit envoyer $p-1$ messages d'informations à ses partenaires. Bien que le nombre de nogoods soit majoré par $O(n^2d^2)$, on se rend compte que le coût des communications peut devenir très important, voire prohibitif. C'est pourquoi nous ajoutons aux p solveurs un processus appelé « gestionnaire de nogoods », dont le rôle entre autres sera d'informer les solveurs des découvertes faites par d'autres solveurs.
Ainsi, quand un solveur découvre un nogood, il en informe aussitôt le gestionnaire qui répandra alors la nouvelle. De cette manière, le solveur n'émet qu'un seul message et retourne plus rapidement à sa tâche première, qui est la résolution du CSP.
Le paragraphe suivant est consacré au rôle et à l'apport du gestionnaire de nogoods.

3.1. *Gestionnaire de nogoods*

3.1.1. *Rôle du gestionnaire*

Le gestionnaire de nogoods a pour mission de mettre à jour la base de nogoods et de communiquer les nogoods découverts aux solveurs. Mettre à jour la base de nogoods revient à ajouter des contraintes au problème initial ou à renforcer les contraintes déjà existantes. A un nogood d'arité 1 (respectivement 2) correspond une contrainte unaire (resp. binaire). Tout nogood communiqué au gestionnaire est ajouté à la base.
Afin de limiter le nombre de communications, on ne souhaite informer les solveurs que des nogoods qui peuvent leur être utiles. Un nogood est dit *utile* pour un solveur s'il permet à ce solveur de limiter la taille de son arbre de recherche.

Le théorème suivant caractérise l'utilité des nogoods en fonction de leur arité et de l'affectation courante.

Théorème 4 (caractérisation de l'utilité des nogoods)

(a) un nogood unaire est toujours utile,

(b) un nogood binaire $(\{x_i \leftarrow a, x_j \leftarrow b\}, J)$ *est utile si, dans l'affectation courante,* x_i *et* x_j *sont affectées respectivement à a et b,*

(c) un nogood binaire $(\{x_i \leftarrow a, x_j \leftarrow b\}, J)$ *peut être utile si, dans l'affectation courante,* x_i *(resp.* x_j*) est affectée à a (resp. b) et que* x_j *(resp.* x_i*) n'est pas encore instanciée.*

Preuve : Voir [TER 01].

A partir de ce théorème, on précise quels sont les solveurs qui doivent recevoir tel nogood (en fonction de son utilité) :

(a) tout nogood unaire est communiqué à tous les solveurs (hormis celui qui l'a découvert),

(b) on communique un nogood binaire $(\{x_i \leftarrow a, x_j \leftarrow b\}, J)$ à tout solveur (hormis celui qui l'a découvert) dont l'affectation contient $x_i \leftarrow a$ ou $x_j \leftarrow b$.

Dans le cas (b), on ne peut pas certifier que le nogood sera utile. En effet, le solveur peut avoir backtracké entre le moment de l'émission du message par le gestionnaire et celui de sa réception par le solveur.
Toujours dans le but de limiter le coût des communications, seule l'affectation $\mathcal{A}$ du nogood $(\mathcal{A}, J)$ est transmise. Communiquer la justification J n'est pas nécessaire car ce nogood vient d'être ajouté au problème sous la forme d'une contrainte c. Grâce à l'information reçue, les solveurs peuvent interdire $\mathcal{A}$ avec pour justification c.

3.1.2. *Apport du gestionnaire*

Dans cette partie, nous mettons en évidence l'avantage apporté par le gestionnaire de nogoods dans notre solveur multiple par rapport à une version de ce solveur sans gestionnaire. La comparaison est basée sur le nombre de messages échangés par tous les processus sur l'ensemble de la résolution.
Soit N le nombre total de nogoods échangés par l'ensemble des solveurs. On comptabilise U nogoods unaires et B binaires. Notons que, parmi ces N nogoods, il peut exister des doublons (deux solveurs pouvant découvrir séparément un même nogood). Dans un schéma sans gestionnaire, chaque solveur communique les nogoods qu'il trouve aux $p-1$ autres solveurs. Globalement, dans ce schéma, $U(p-1)$ messages sont émis pour les nogoods unaires et $B(p-1)$ pour les binaires.
Dans un schéma avec gestionnaire, les nogoods sont d'abord transmis par les solveurs au gestionnaire. Sur l'ensemble de la recherche, les solveurs envoient au gestionnaire U messages pour les nogoods unaires et B pour les binaires. Le gestionnaire envoie alors u nogoods unaires aux $p-1$ solveurs. Ces u nogoods correspondent aux U nogoods desquels on a retiré les doublons. De même, pour les nogoods binaires, les doublons ne sont pas communiqués. De plus, les nogoods binaires qui restent ne sont

pas systématiquement communiqués à tous les autres solveurs, le gestionnaire restreignant le nombre de destinataires. Soit b le nombre total de messages envoyés par le gestionnaire pour les nogoods binaires. Globalement, dans le schéma avec gestionnaire, on émet $U + u(p-1)$ messages pour les nogoods unaires et $B + b$ messages pour les binaires.

Dans le pire des cas, le schéma avec gestionnaire produit jusqu'à N messages supplémentaires par rapport au schéma sans gestionnaire. Mais, en général, nous avons observé que u et b étaient suffisamment petits pour que le schéma avec gestionnaire utilise moins de messages.

3.2. *Phase d'interprétation*

La phase d'interprétation que nous allons décrire est appliquée à chaque fois qu'un nogood parvient au solveur. Pour information, on teste si un message a été reçu après chaque développement d'un nœud et avant de réaliser le filtrage.
Dans la phase d'interprétation, les solveurs analysent les nogoods reçus afin de limiter la taille de leur arbre de recherche en stoppant le développement d'affectations menant à des échecs ou en appliquant un filtrage supplémentaire. Pour les nogoods unaires, cette phase se traduit par une suppression définitive d'une valeur et éventuellement par un retour en arrière. La méthode 1 détaille cette phase pour de tels nogoods :

Méthode 1 (phase d'interprétation pour les nogoods d'arité 1)
Soit $\mathcal{A}$ l'affectation courante. Soit ($\{x_i \leftarrow a\}$,J) le nogood reçu par le solveur.
On supprime a de D_i.

(a) Si x_i est affectée à la valeur a, alors on backjumpe jusqu'à la variable x_i.
Si D_i est vide, on mémorise le nogood ($\mathcal{A}[X_K]$,K), avec K l'ensemble des value-killers de D_i.

(b) Si x_i est affectée à une valeur différente de a, alors on ne fait rien.

(c) Si x_i n'est pas affectée, alors on vérifie si D_i est vide.
Si D_i est vide, on mémorise le nogood ($\mathcal{A}[X_K]$,K), avec K l'ensemble des value-killers de D_i.

Théorème 5 *La phase d'interprétation pour les nogoods d'arité 1 est correcte.*

Preuve : Voir [TER 01].

Pour les nogoods binaires, la phase revient à appliquer un filtrage supplémentaire et éventuellement à effectuer un retour en arrière. La méthode 2 décrit les actions réalisées dans cette phase pour des nogoods binaires.

Méthode 2 (phase d'interprétation pour les nogoods d'arité 2)
Soient $\mathcal{A}$ l'affectation courante et ($\{x_i \leftarrow a$,$x_j \leftarrow b\}$,J) le nogood reçu par le solveur.

(a) Si x_i et x_j sont affectées dans $\mathcal{A}$ à a et b respectivement, alors on revient en arrière sur la variable la plus profonde parmi x_i et x_j.

Si x_j (resp. x_i) est cette variable, on supprime par filtrage b (resp. a) de D_j (resp. D_i).

(b) Si x_i (resp. x_j) est affectée à a (resp. b) et que x_j (resp. x_i) n'est pas instanciée, alors on supprime par filtrage b (resp. a) de D_j (resp. D_i).

Si D_j (resp. D_i) est vide, on mémorise le nogood $(\mathcal{A}[X_K],K)$ avec K l'ensemble des value-killers de D_j (resp. D_i).

Contrairement à la phase d'interprétation pour les nogoods unaires, la suppression ici n'est pas définitive.

Théorème 6 *La phase d'interprétation pour les nogoods d'arité 2 est correcte.*

Preuve : Voir [TER 01].

Bien que la phase d'interprétation soit correcte, son intégration à l'algorithme FC-NR peut, dans certains cas, compromettre une propriété de base de FC-NR. Dans le paragraphe suivant, après avoir rappelé cette propriété, nous décrivons ce problème et nous proposons des solutions.

3.3. *Maintien de la FC-consistance*

Nous rappelons d'abord une propriété de base de FC-NR (héritée de FC) :

Propriété 1 *Toute affectation construite par FC-NR est FC-consistante.*

Lors d'un retour en arrière sur une variable x_i, FC-NR (comme FC) annule l'effet du filtrage consécutif à l'affectation de x_i. Elle restaure donc chaque domaine dans l'état dans lequel il se trouvait avant le filtrage. En particulier, si un domaine devient vide à l'issue d'un filtrage, il ne l'est plus après restauration. Il en découle la conservation de la propriété 1.
C'est la conservation de cette propriété qui est mise en péril par l'ajout de la phase d'interprétation. Pour illustrer ce phénomène, nous considérons l'arbre de recherche parcouru par un solveur travaillant en coopération avec d'autres. Soit $D_4 = \{a,b,c,d\}$. Ce solveur instancie d'abord x_1 à a, puis x_2 à b. Le filtrage consécutif à l'affectation de x_2 supprime a de D_4. Celui qui suit l'affectation de x_3 à c_1 retire b et c de D_4. Le solveur affecte x_4 à la valeur d, puis il développe le sous-arbre correspondant. Supposons que ce sous-arbre ne contienne que des échecs et que le solveur ait reçu des nogoods unaires interdisant entre autres l'affectation de x_4 avec les valeurs b et c. Le sous-arbre ne conduisant pas à une solution, le solveur revient en arrière et mémorise un nogood unaire interdisant la valeur d pour x_4. Il revient alors à la variable x_3 et instancie x_3 avec c_2. Mais, un problème se pose : D_4 est vide (à cause des suppressions définitives de b, c et d de D_4 par des nogoods unaires). Donc, l'affectation construite

n'est pas FC-consistante et la propriété 1 n'est plus conservée.
Le théorème suivant permet de caractériser le problème :

Théorème 7
Soit une affectation FC-consistante $\mathcal{A}_j = \{x_1 \leftarrow v_1, \ldots, x_{j-1} \leftarrow v_{j-1}, x_j \leftarrow r\}$.
On considère l'exploration du sous-arbre de racine $x_j \leftarrow r$ *par l'algorithme FC-NR. Soit* NG *l'ensemble des valeurs de* x_i *qui demeurent interdites par des nogoods à l'issue de cette exploration tels que les nogoods considérés soient mémorisés ou reçus pendant cette exploration et qu'aucun d'entre eux n'implique* x_j.
Si toutes les valeurs de $D_i(\mathcal{A}_j)$ *ont été supprimées lors de cette exploration, aucune valeur n'est restaurée dans* $D_i(\mathcal{A}_{j-1})$ *après l'annulation du filtrage consécutif à l'affectation de* x_j *à* r *si et seulement si* $D_i(\mathcal{A}_{j-1}) \subseteq NG$.

Preuve : Voir [TER 01].

Il en résulte que, dans certains cas, la conjonction de la réception et de la création de nogoods avec le filtrage provoque l'existence de domaine vide après annulation du filtrage et, ainsi, la perte de la propriété 1.

Une parade à ce problème consisterait donc à vérifier l'existence d'un domaine vide après l'annulation du dernier filtrage, et si un domaine est vide, à backtracker jusqu'à ce qu'il ne le soit plus. Il n'est pas nécessaire de tester la vacuité de tous les domaines, grâce au lemme suivant qui précise les causes potentielles du problème.

Lemme 1 *Seule la mémorisation ou la réception d'un nogood peut entraîner la perte de la propriété 1.*

Preuve : Voir [TER 01].

Ce lemme permet de ne tester que la vacuité des domaines devenus vides après réception ou mémorisation d'un nogood. Dans le cas d'une vacuité provoquée par la réception d'un nogood, nous introduisons une phase de backjump supplémentaire. Cette phase suit toute découverte de domaine vide lors de la réception d'un nogood et permet de poursuivre la recherche à partir d'une affectation FC-consistante.

Méthode 3 (phase de backjump)
Si D_i *devient vide après la réception d'un nogood, alors on revient en arrière :*

- *jusqu'à ce que* D_i *soit non vide ou jusqu'à ce qu'on retourne à l'affectation vide, si le nogood est unaire,*
- *jusqu'à ce que* D_i *soit non vide s'il est binaire.*

Le retour à l'affectation vide n'a lieu que dans le cas de problèmes inconsistants.

Théorème 8 *La phase de backjump décrite par la méthode 3 est correcte.*

Preuve : Voir [TER 01].

Dans le cas d'une vacuité provoquée par la mémorisation d'un nogood, on ne mémorise que les nogoods qui ne rendent pas vide un domaine. On communique toutefois

tous les nogoods découverts qu'ils rendent vide ou non un domaine. L'inconvénient de cette solution est qu'on ne tire pas pleinement partie de tous les nogoods découverts. Toutefois, elle possède l'avantage d'une mise en œuvre facile. De plus, elle produit de meilleurs résultats par rapport aux autres solutions (comme une phase de backjump supplémentaire similaire à celle de la méthode 3).

4. Résultats expérimentaux

4.1. *Protocole expérimental*

Nous travaillons sur des instances aléatoires produites par le générateur aléatoire écrit par D. Frost, C. Bessière, R. Dechter et J.-C. Régin. Ce générateur[1] prend 4 paramètres N, D, C et T. Il construit un CSP de la classe (N,D,C,T) avec N variables dont les domaines sont de taille D et C contraintes binaires ($0 \leq C \leq \frac{N(N-1)}{2}$) dans lesquelles T n-uplets sont interdits ($0 \leq T \leq D^2$).

Les résultats expérimentaux donnés par la suite concernent des problèmes des classes $(50,25,123,T)$ avec T qui varie entre 433 et 447. Les classes considérées sont voisines du seuil de satisfaisabilité qui se situe à $T = 437$. Toutefois, pour l'algorithme FC-NR, le pic de difficulté est situé à $T = 439$. Tous les problèmes considérés possèdent un graphe de contraintes connexe.

Les résultats donnés correspondent aux moyennes des résultats obtenus sur 100 problèmes par classe. Chaque problème est résolu 15 fois afin de réduire l'impact du non-déterminisme des solveurs sur les résultats. Les résultats d'un problème sont alors les moyennes des résultats des 15 résolutions. Pour une résolution donnée, les résultats pris en compte sont ceux du premier solveur qui résout le problème.

Pour information, les expériences sont réalisées sur un PC sous Linux équipé d'un processeur Pentium III 550 MHz d'Intel, et doté de 256 Mo de mémoire.

4.2. *Heuristiques*

Nous devons garantir l'existence d'un arbre de recherche distinct pour chaque solveur. Pour atteindre ce but, chaque solveur possède une heuristique différente pour choisir les variables et/ou les valeurs. Le principal problème est de produire suffisamment d'heuristiques efficaces. Hélas, les heuristiques efficaces sont peu nombreuses. Par contre, à partir d'une heuristique donnée, on peut décliner plusieurs ordres distincts en choisissant différemment la première variable et en employant l'heuristique ensuite. C'est la solution que nous avons adoptée.

Nous utilisons comme principale heuristique pour le choix des variables l'heuristique dom/deg ([BES 96]) pour laquelle la prochaine variable à instancier est celle

1. on peut le télécharger à l'adresse http://www.lirmm.fr/~bessiere/generator.html.

qui minimise le rapport $\frac{|D_i|}{|\Gamma_i|}$ (avec D_i le domaine courant de la variable x_i et Γ_i l'ensemble des sommets connectés à x_i par une contrainte binaire).
Cette heuristique est considérée comme meilleure, en général, que les autres heuristiques classiques sur les CSPs, ce qui explique son choix.
Dans notre implémentation, seule la taille des domaines varie à chaque affectation (lors du filtrage). Le degré n'est incrémenté que lorsqu'une nouvelle contrainte est ajoutée grâce à un nogood.

En ce qui concerne le choix de la prochaine valeur à affecter, on considère les valeurs dans leur ordre d'apparition ou dans l'ordre inverse. Dans les résultats qui suivent (sauf indication contraire), une moitié des solveurs utilise l'ordre d'apparition des valeurs pour ordonner les domaines, l'autre moitié l'ordre inverse.

4.3. ***Résultats***

4.3.1. *Efficacité*

Dans ce paragraphe, nous estimons l'accélération et l'efficacité de notre méthode. Si T_1 est le temps nécessaire à un solveur pour résoudre une série de problèmes et T_p le temps de résolution pour p solveurs lancés en parallèle, on définit l'accélération comme le rapport $\frac{T_1}{T_p}$ et l'efficacité comme le rapport $\frac{T_1}{p\ T_p}$. L'accélération est dite linéaire par rapport au nombre p de solveurs si elle est égale à p, superlinéaire si elle est supérieure à p, sublinéaire sinon.

T	# problèmes consistants	p							
		2	4	6	8	10	12	14	16
433	76	1.52	1.45	1.35	1.28	1.11	0.98	0.92	0.87
434	70	1.34	1.43	1.36	1.37	1.29	1.24	1.13	0.97
435	66	1.44	1.46	1.32	1.34	1.22	1.22	1.11	1.06
436	62	1.58	1.38	1.36	1.19	1.10	1.01	0.97	0.91
437	61	1.14	1.38	1.36	1.26	1.20	1.09	1.02	0.90
438	37	1.40	1.53	1.50	1.35	1.34	1.20	1.07	0.96
439	39	1.30	1.28	1.13	1.21	1.07	0.97	0.88	0.79
440	31	1.25	1.28	1.15	1.10	1.00	0.94	0.90	0.82
441	25	1.35	1.33	1.32	1.23	1.13	1.02	0.94	0.87
442	13	1.06	1.37	1.24	1.19	1.06	0.97	0.85	0.76
443	10	1.47	1.32	1.20	1.14	1.00	0.90	0.83	0.76
444	8	1.36	1.38	1.27	1.18	1.08	0.98	0.89	0.77
445	3	1.40	1.23	1.17	1.08	0.99	0.90	0.81	0.74
446	2	1.26	1.27	1.17	1.08	0.99	0.89	0.79	0.73
447	3	1.44	1.33	1.17	1.07	0.94	0.84	0.75	0.66

Tableau 1. *Efficacité pour des problèmes consistants et inconsistants des classes* $(50{,}25{,}123{,}T)$.

Le tableau 1 présente l'efficacité obtenue pour des problèmes consistants et inconsistants des classes $(50{,}25{,}123{,}T)$ avec T qui varie entre 433 et 447. Dans le tableau 1,

nous observons une accélération linéaire ou superlinéaire sur toutes les classes (hormis 3 classes) jusqu'à 10 solveurs. Au delà de 10 solveurs, les résultats sont plus mitigés avec quelques classes où l'accélération reste linéaire ou superlinéaire et surtout avec une majorité de classes où elle devient sublinéaire. Nous notons aussi une diminution de l'efficacité avec l'augmentation du nombre de solveurs.
Si on distingue les problèmes suivant leur consistance, on observe que notre méthode est, en général, plus efficace sur les problèmes consistants (l'accélération restant linéaire ou superlinéaire). Pour les problèmes consistants, comme pour les inconsistants, on note une diminution de l'efficacité. Mais, l'efficacité sur les problèmes inconsistants est moins importante, bien qu'elle soit supérieure ou égale à 1 jusqu'à 10 solveurs. Il en résulte l'apparition de classes sur lesquelles notre méthode a une accélération sublinéaire à partir de 10 solveurs. C'est donc cette perte d'efficacité sur les problèmes inconsistants qui entraîne une diminution des performances sur l'ensemble des problèmes (consistants et inconsistants).

4.3.2. *Origines des résultats obtenus*

D'abord, nous nous intéressons aux origines des gains. Il existe deux origines possibles : la concurrence et la coopération. Nous déclinons donc notre schéma en une version concurrente (c'est-à-dire dépourvue d'échange de nogoods). Nous comparons les deux versions en effectuant le rapport du temps de résolution pour la version concurrente par celui de la version coopérative. Les résultats obtenus pour un nombre de solveurs compris entre 2 et 8 solveurs sont représentés par les courbes de la figure 1, pour la classe (50,25,123,439). Ces résultats sont représentatifs de ceux observés sur les autres classes.

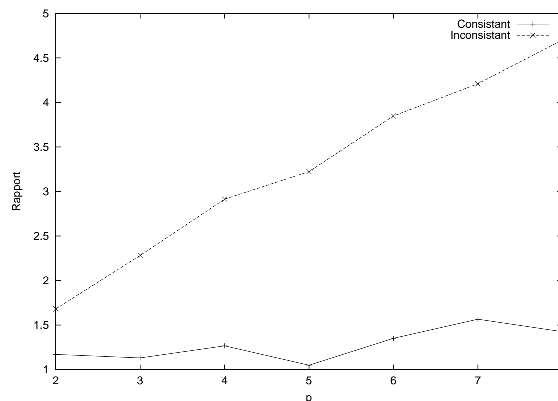

Figure 1. *Rapport concurrence / coopération pour la classe* (50,25,123,439).

On constate d'abord que la version coopérative est toujours meilleure que la version concurrente. Ensuite, on note que le rapport est assez proche de 1 pour les problèmes consistants. Cela signifie que la qualité des résultats, pour ces problèmes, provient

en grande partie de la concurrence. Cependant, si le rapport est proche de 1, il reste toujours supérieur à 1, ce qui implique que l'échange de nogoods participe également à la qualité des résultats obtenus.
Enfin, pour les problèmes inconsistants, on observe que le rapport est plus important que pour les problèmes consistants et qu'il augmente avec le nombre de solveurs. Autrement dit, les résultats obtenus sur ces problèmes proviennent en majeure partie de la coopération et l'apport de la coopération augmente avec le nombre de solveurs.

Pour les problèmes inconsistants, il faut aussi souligner le rôle prépondérant des heuristiques de choix de valeurs employées. Pour chaque solveur s (sauf un si le nombre de solveurs est impair), il existe un solveur qui utilise la même heuristique pour le choix des variables que s et l'heuristique inverse de celle de s pour le choix des valeurs. Sans échange de nogoods, ces deux solveurs parcourent des arbres de recherche voisins. Avec l'échange de nogoods, chacun ne visite plus qu'une partie de leur arbre, grâce en partie aux nogoods trouvés par l'autre. Il en est de même pour les problèmes consistants, mais l'effet est moins visible car la recherche s'arrête dès qu'une solution est trouvée.

Nous nous focalisons sur les causes possibles de la diminution de l'efficacité. Avec un schéma comme le nôtre, une cause courante de la perte d'efficacité est l'importance du coût des communications. Notre méthode ne fait pas exception. Mais, dans notre cas, il existe une autre cause qui explique la chute des performances.
Nous comparons les solveurs multiples S_8 et S_8'. Les deux ont 8 solveurs. Pour ordonner les domaines, la moitié des solveurs de S_8 utilise l'ordre d'apparition des valeurs, et l'autre moitié l'ordre inverse. Tous les solveurs de S_8' emploient l'ordre d'apparition des valeurs. On se rend compte que S_8 possède une meilleure efficacité que S_8'. Certes, le nombre de messages échangés dans S_8' est plus important que dans S_8. Mais, surtout, il en est de même du nombre de nœuds. Cela signifie que S_8' explore des arbres plus importants. Les méthodes S_8 et S_8' se distinguent par les heuristiques utilisées pour les choix de valeurs et de variables. Les heuristiques employées pour le choix des variables sont assez proches les unes des autres. Utiliser deux ordres distincts pour les valeurs ajoute de la diversité aux résolutions. Ainsi, S_8 a une plus grande diversité que S_8'. Cette différence de diversité permet en partie d'expliquer l'écart de performances entre S_8 et S_8'.
Le manque de diversité est une des principales causes (avec l'augmentation du nombre des communications) de la diminition de l'efficacité avec le nombre de solveurs.

4.3.3. *Nombres de messages et apport réel du gestionnaire*

Dans ce paragraphe, nous étudions l'apport réel du gestionnaire dans notre schéma. Pour mesurer cet apport, nous comparons les coûts engendrés par les communications dans un schéma avec gestionnaire et dans un schéma sans gestionnaire.
Dans les résultats présentés, nous considérons que le coût d'un message est indépendant de la présence ou non du gestionnaire de nogoods, et que la communication d'un nogood binaire est deux fois plus coûteuse que celle d'un nogood unaire. La seconde hypothèse se justifie par le fait qu'un nogood binaire est formé de deux couples

variable-valeur, contre un seul pour un nogood unaire. Il faut donc transmettre deux fois plus de données. La figure 2 présente le rapport des coûts des communications

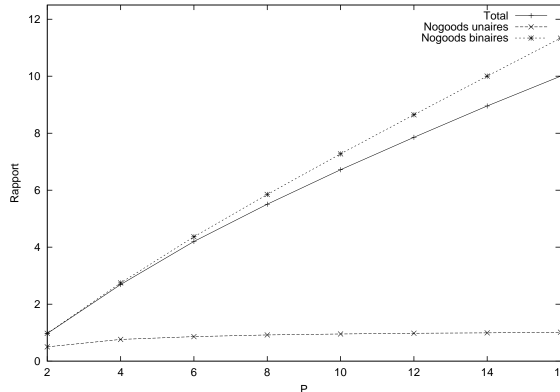

Figure 2. *Rapport des coûts des communications entre les schémas sans et avec gestionnaire de nogoods.*

entre un schéma sans gestionnaire et un avec gestionnaire. Les problèmes considérés (consistants et inconsistants) appartiennent à la classe (50,25,123,439). Pour information, les observations réalisées en utilisant d'autres classes de problèmes sont similaires. Il en est de même si on distingue les problèmes suivant leur consistance.
Globalement, on constate que l'apport du gestionnaire augmente avec le nombre de solveurs. Ainsi, on peut espérer que le nombre de solveurs à partir duquel le coût des communications pénalise l'efficacité soit plus important dans un schéma avec gestionnaire que dans un schéma sans gestionnaire.
Plus précisément, on note d'abord que le coût des communications des nogoods unaires est plus important pour le schéma avec gestionnaire, quand le nombre de solveurs est peu important. Ce résultat était prévisible. En effet, dans le cas des nogoods unaires, le gestionnaire n'élimine que les doublons. Or, la probabilité que deux solveurs trouvent simultanément le même nogood unaire est d'autant plus faible que le nombre de solveurs est petit. On explique ainsi le fait que le schéma avec gestionnaire devient meilleur que celui sans gestionnaire quand le nombre de solveurs augmente.
En ce qui concerne le coût des communications des nogoods binaires, le schéma avec gestionnaire est nettement plus économique et cette économie augmente avec le nombre de solveurs. Ce résultat s'explique par le fait que les nogoods binaires ne sont, en général, utiles qu'à un petit nombre de solveurs.
Sur l'ensemble des communications, le schéma avec gestionnaire est le meilleur. Ce résultat est dû essentiellement au nombre de nogoods binaires qui est nettement supérieur à celui des nogoods unaires (d'un facteur compris entre 30 et 100).

En conclusion, le gestionnaire remplit donc parfaitement son rôle en limitant le nombre de messages échangés. Il évite ainsi aux solveurs une perte de temps dans la

gestion des communications (en particulier dans la réception de nogoods inutiles). Il leur permet donc de se consacrer pleinement à la résolution du problème.

5. Conclusions

Dans ce papier, nous définissons un nouveau schéma de recherche parallèle coopérative avec échange de nogoods et nous estimons expérimentalement son efficacité. Nous observons alors des accélérations linéaires ou superlinéaires jusqu'à 10 solveurs pour les problèmes inconsistants, et jusqu'à 16 solveurs pour ceux consistants. Donc, l'échange de nogoods est une forme efficace de coopération. Nous notons également une diminution de l'efficacité avec l'augmentation du nombre de solveurs. Cette diminution, nous l'attribuons au nombre croissant des communications et au manque de diversité des solveurs.

Une première extension de ce travail consiste à trouver des heuristiques efficaces et suffisamment diverses pour améliorer l'efficacité ainsi que pour augmenter le nombre de solveurs. Ensuite, nous pouvons étendre notre schéma en appliquant tout algorithme qui maintient un certain niveau de consistance, en utilisant plusieurs algorithmes (ce qui permettrait de mêler des recherches complètes et incomplètes à l'image de [HOG 93b] et ainsi que d'augmenter la diversité des solveurs), ou en le généralisant à d'autre forme de coopération avec échange d'informations.

6. Bibliographie

[BES 96] BESSIÈRE C., RÉGIN J.-C., « MAC and Combined Heuristics : Two Reasons to Forsake FC (and CBJ?) on Hard Problems », *Proc. of CP 96*, 1996, p. 61-75.

[CLE 91] CLEARWATER S., HUBERMAN B., HOGG T., « Cooperative Solution of Constraint Satisfaction Problems », *Science*, vol. 254, 1991, p. 1181-1183.

[HOG 93a] HOGG T., HUBERMAN B. A., « Better Than the Best: The Power of Cooperation », NADEL L., STEIN D., Eds., *1992 Lectures in Complex Systems*, vol. V de *SFI Studies in the Sciences of Complexity*, p. 165-184, Addison-Wesley, 1993.

[HOG 93b] HOGG T., WILLIAMS C., « Solving the Really Hard Problems with Cooperative Search », *Proc. of AAAI 93*, 1993, p. 231-236.

[HOG 94] HOGG T., WILLIAMS C., « Expected Gains from Parallelizing Constraint Solving for Hard Problems », *Proc. of AAAI 94*, Seattle, WA, 1994, p. 331-336.

[MAR 96] MARTINEZ D., VERFAILLIE G., « Echange de Nogoods pour la résolution coopérative de problèmes de satisfaction de contraintes », *Actes de JNPC 96*, 1996, p. 261-274.

[SCH 93] SCHIEX T., VERFAILLIE G., « Nogood Recording for Static and Dynamic Constraint Satisfaction Problems », *Proc. of the 5th IEEE ICTAI*, 1993.

[TER 01] TERRIOUX C., « Recherche Coopérative et Nogood Recording », Rapport de Recherche, LIM, 2001.