Un algorithme de résolution d'instances CSP s'appuyant sur les variables FAC

Éric Grégoire **Jean-Marie Lagniez** **Bertrand Mazure**

Université Lille-Nord de France
CRIL - CNRS UMR 8188
Artois, F-62307 Lens

{gregoire,lagniez,mazure}@cril.fr

Résumé

La contribution de ce papier est double. D'une part, il introduit le concept de variable FAC dans le cadre du problème de satisfaction de contraintes (CSP). Ces variables FAC, découvertes à l'aide de la recherche locale, permettent d'augmenter l'efficacité des approches de type MAC. D'autre part, une nouvelle combinaison entre la recherche locale, l'arc-consistance généralisé et un solveur de type MAC permettant d'améliorer la synergie de ces trois approches est proposée. En créant des flux multidirectionnels d'informations entre les différents composants de recherche, cette technique s'avère plus compétitive que les techniques usuelles sur la plupart des classes d'instances.

Abstract

The contribution of this paper is twofold. On the one hand, it introduces a concept of FAC variables in discrete Constraint Satisfaction Problems (CSPs). FAC variables can be discovered by local search techniques and powerfully exploited by MAC-based methods. On the other hand, a novel synergetic combination schema between local search paradigms, generalized arc-consistency and MAC-based algorithms is presented. By orchestrating a multiple-way flow of information between these various fully integrated search components, it proves more competitive than the usual techniques on most classes of instances.

1 Introduction

Le formalisme CSP (*Constraint Satisfaction Problem*) constitue un cadre puissant pour la représentation et la résolution efficace de nombreux problèmes. En particulier, de nombreux problèmes académiques ou réels peuvent être formulés dans ce cadre. À l'heure actuelle, les CSP sont généralement résolus par différentes version d'algorithmes de recherche systématique de type « backtrack ». Une autre possibilité, pour les résoudre, est d'utiliser une méthode de recherche locale (RL) qui est généralement basée sur le concept *min-conflicts* introduit et développé par Minton *et al.* [22]. Bien qu'étant incomplets, c'est-à-dire qu'ils ne fournissent aucune garantie de trouver une solution ou de prouver l'incohérence du problème, ces algorithmes sont souvent efficaces pour trouver une solution. Au vu de son efficacité, elle fut pendant longtemps considérée comme l'approche la plus puissante pour la résolution du problème CSP (et de son homologue SAT) [11, 25, 4].

Cependant, à part dans certains cas spécifiques (*e.g.* [8], [14]), l'approche principalement utilisée pour la résolution de CSP est de type recherche en profondeur d'abord avec retour arrière, et n'inclut généralement pas la recherche locale comme composante principale (*e.g.* Abscon [21, 17], Choco [27], Mistral [13], Sugar [26], etc.). Une des raisons est que la RL n'est pas une méthode complète et n'est pas capable de prouver l'insatisfiabilité d'une formule. De plus, la RL n'utilise aucune des techniques de filtrage (*e.g.* arc-consistance) habituellement employées dans le cadre des approches complètes et qui permettent d'éviter l'exploration d'espaces de recherche non pertinents. Enfin, il est parfois (mais à tort) estimé que la RL devrait simplement être dédiée à la résolution de problèmes où les solutions sont densément réparties dans l'espace de recherche, justifiant ainsi la possibilité d'un certain aspect aléatoire de l'instance considérée.

Ce papier montre que les techniques complètes et la RL peuvent s'avérer complémentaires pour permettre de ré-

soudre le problème CSP. Plus précisément, il présente une combinaison synergique de la recherche locale et d'éléments de techniques complètes, qui souvent surpassent les approches complètes usuelles. Cette méthode n'est pas seulement complète : elle est aussi robuste dans le sens où elle peut aussi bien résoudre les instances satisfiables et insatisfiables, structurées ou aléatoires. En effet notre étude expérimentale montre qu'elle résout globalement plus d'instances que les techniques utilisées couramment.

Une des clés de notre approche est que la recherche locale permet d'extraire les informations nécessaires afin de guider autant que possible la recherche vers les sous-parties difficiles du problème CSP considéré. Bien que cette idée ait déjà été exploitée dans le domaine de SAT [19], elle est ici raffinée grâce au concept original de variables FAC. Ces dernières, dans le cadre CSP, sont des variables qui apparaissent dans toutes les contraintes violées pour une certaine affectation de l'ensemble des variables. Ces variables apparaissent donc dans au moins une contrainte par noyau minimalement incohérent (aussi appelé MUC pour *Minimal Unsatisfiable Core*) d'un CSP. De manière intéressante, la RL permet souvent de détecter efficacement des variables FAC, et permet à une approche complète de type MAC de se focaliser en priorité sur ces variables. De cette manière, les résultats sont significativement améliorés sur de nombreuses instances. De même, des heuristiques puissantes (en particulier $dom/wdeg$ [3]) développées dans le cadre des approches complètes utilisées en CSP peuvent jouer un rôle essentiel dans le processus de RL. Ainsi la méthode proposée, appelée FAC-SOLVER, correspond à la combinaison élaborée de la RL et d'étapes issues de méthodes complètes. Ceci est permis par des échanges réciproques d'informations entre les différentes composantes mises en jeu.

Ce papier est organisé de la manière suivante. Dans la prochaine section les définitions, notations et notions préliminaires sont introduites. Ensuite, le concept de variables FAC est présenté. Dans la section 4, l'architecture de notre approche hybride est détaillée. Avant de conclure et de donner quelques perspectives, l'ensemble des expérimentations conduites est discuté.

2 Définitions, notations et notions préliminaires

Un CSP ou réseau de contraintes $\mathcal{CN}$ est un couple $\langle \mathcal{X}, \mathcal{C} \rangle$ où $\mathcal{X}$ est un ensemble fini de n variables tel qu'à chaque variable X de $\mathcal{X}$ est associé un ensemble fini $dom(X)$ de valeurs. $\mathcal{C}$ est un ensemble fini de m contraintes tel qu'à chaque contrainte C est associée une relation $rel(C)$ indiquant l'ensemble des couples de valeurs autorisés pour les variables $var(C) \subseteq \mathcal{X}$ impliquées par la contrainte C. Une interprétation $\mathcal{I}$ du $\mathcal{CN}$ associe une valeur $\mathcal{I}(X) \in dom(X)$ pour tout $X \in \mathcal{X}$.

Dans la suite, on notera $false(\mathcal{X}, \mathcal{C}, \mathcal{I})$ l'ensemble des variables appartenant à au moins une contrainte violée par l'interprétation $\mathcal{I}$. $\langle \mathcal{X}, \mathcal{C} \rangle|_{X=v}$ est le CSP obtenu lorsque le domaine de la variable X est réduit au singleton $\{v\}$ tandis que $\langle \mathcal{X}, \mathcal{C} \rangle|_{X \neq v}$ est le CSP obtenu en supprimant la valeur v de $dom(X)$. Une interprétation $\mathcal{I}$ se trouve dans un minimum local pour un $\mathcal{CN}$ lorsque $\mathcal{I}$ viole au moins une contrainte du réseau et que quelle que soit l'interprétation voisine $\mathcal{I}'$, c'est-à-dire où une seule valeur est modifiée par rapport à $\mathcal{I}$, le nombre de contraintes violées par $\mathcal{I}'$ ne diminue pas. Une interprétation $\mathcal{I}'$ est voisine de $\mathcal{I}$ si $\exists! X \in \mathcal{X}$ tel que $\mathcal{I}(X) \neq \mathcal{I}'(X)$.

Le problème de satisfaction de contraintes consiste à vérifier si un $\mathcal{CN}$ admet au moins une affectation de ses variables qui satisfait l'ensemble des contraintes du réseau.

L'un des algorithmes de recherche systématique parmi les plus populaires pour résoudre un CSP est MAC (*Maintaining Arc Consistency*) [23]. Cette technique effectue une recherche en profondeur d'abord avec retours-arrière. De plus, à chaque point de choix une consistance locale (AC pour Arc Cohérence) est maintenue. Celle-ci consiste à filtrer les valeurs interdites [18, 2, 16]). Dans la suite de cet article, nous n'imposons aucune limitation à l'arité des contraintes.

3 Les variables FAC

Une des clés de l'efficacité de notre approche est liée au concept de variables FAC présenté ici.

Définition 1. *Soient un réseau de contraintes $\mathcal{CN}$ et une interprétation complète $\mathcal{I}$. Une variable* FAC *est une variable apparaissant dans toutes les contraintes du $\mathcal{CN}$ violées par $\mathcal{I}$.*

Ce concept peut-être rapproché de la notion de *boundary point* introduite par Goldberg dans le cadre du problème SAT [9]. En effet, dans le cadre de SAT, une variable x est considérée comme *boundary* s'il existe une interprétation complète $\mathcal{I}$ construite sur l'ensemble des variables de la formule telle que x appartient à toutes les contraintes falsifiées par $\mathcal{I}$ (ce qui est typiquement la définition d'une variable FAC). Ici, nous avons choisi de ne pas nommer nos variables *boundary* en raison de la nuance suivante : dans le cadre du problème SAT, lorsqu'une variable x est détectée *boundary* pour l'interprétation $\mathcal{I}$, il est possible de satisfaire l'ensemble des contraintes falsifiées par $\mathcal{I}$ en inversant la valeur de vérité de la variable x. Cette propriété permet d'établir qu'une variable *boundary* est à la frontière entre la satisfiabilité et l'insatisfiabilité de la formule. Dans le cadre de CSP, lorsqu'une variable est détectée comme FAC, il n'est pas certain qu'échanger la valeur de cette variable satisfasse la contrainte. Ainsi, la notion de frontière mise en avant par Goldberg ne peut être appliquée ici. C'est pour cela que nous avons choisi

d'utiliser des termes différents.

Les propriétés suivantes peuvent aider à comprendre le rôle potentiel des variables FAC pour la résolution du problème CSP.

Propriété 1. *Soit un réseau de contraintes $\mathcal{CN}$ insatisfiable, si X est une variable* FAC *alors X apparaît dans au moins une contrainte de chaque* MUC *du $\mathcal{CN}$.*

Démonstration. Quelle que soit l'interprétation complète $\mathcal{I}$, au moins une contrainte de chaque MUC est violée par $\mathcal{I}$ (par définition des MUC). Ainsi, si la variable X a été détectée FAC à l'aide de l'interprétation $\mathcal{I}$, alors par définition d'une variable FAC, X apparaît dans toutes les contraintes violées par $\mathcal{I}$. Donc X appartient à au moins une contrainte de chaque MUC. □

Puisqu'elles apparaissent dans tous les MUC, les variables FAC peuvent être considérées comme très intéressantes pour conduire à l'établissement de la preuve de l'incohérence d'un réseau. Cependant établir si une variable est FAC est calculatoirement difficile. En effet, le problème qui consiste à vérifier si une contrainte donnée appartient à au moins un MUC est Σ_2^p [7]. De plus, un CSP peut posséder un nombre exponentiel de MUC. Ce qui implique que, contrôler si une variable est FAC en calculant l'ensemble des MUC du problème est irréalisable dans le pire des cas. Pour pallier ce problème, nous proposons d'utiliser la recherche locale afin de détecter des variables FAC. Pour cela, une méthode de recherche locale classique est lancée. Lorsqu'un minimum local est atteint, l'ensemble des contraintes violées par cette interprétation est parcouru à la recherche de variables FAC. Le fait d'effectuer cette recherche lorsqu'un minimum local est atteint n'est pas fortuit. En effet, il semble raisonnable de ne pas ralentir les performances du solveur en vérifiant chacune des interprétations parcourues par la recherche locale.

Propriété 2. *Un réseau de contraintes $\mathcal{CN}$ qui contient au moins deux* MUC *disjoints (n'ayant aucune variable commune) ne possède pas de variable* FAC.

Démonstration. Soient $\mathcal{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$ un réseau de contraintes, $\mathcal{M}_1 = \langle \mathcal{X}_1, \mathcal{C}_1 \rangle$ et $\mathcal{M}_2 = \langle \mathcal{X}_2, \mathcal{C}_2 \rangle$ deux MUC du $\mathcal{CN}$ tel que $(\mathcal{X}_1 \cap \mathcal{X}_2) = \emptyset$. Raisonnons par l'absurde, supposons qu'il existe $X \in \mathcal{X}$ une variable FAC pour une certaine interprétation $\mathcal{I}$. Par définition des MUC, il existe $C_1 \in \mathcal{C}_1$ et $C_2 \in \mathcal{C}_2$ deux contraintes falsifiées par $\mathcal{I}$. Puisque X est une variable FAC pour $\mathcal{I}$, X appartient à toutes les contraintes falsifiées par $\mathcal{I}$. Donc $X \in C_1$ et $X \in C_2$, ce qui est absurde puisque $(\mathcal{X}_1 \cap \mathcal{X}_2) = \emptyset$. □

La propriété 2 met en avant le fait qu'il n'est pas toujours possible de trouver des variables FAC. Cependant cette situation est rare, puisque dans la plupart des cas les instances insatisfiables ne possèdent pas (ou très rarement) de noyaux inconsistants disjoints, synonyme de deux sources différentes d'incohérence.

Finalement, nous pouvons noter que, dans le cadre d'instances satisfiables, il est tout de même possible de détecter des variables FAC. Ces variables peuvent jouer un rôle important dans la recherche d'une solution en mettant en exergue les parties difficiles du problème. De plus, lors d'un processus de recherche classique (MAC), l'affectation d'une variable peut conduire à obtenir un CSP inconsistant. La détection de variables FAC peut, dans ce contexte, aider à réfuter plus rapidement cette mauvaise décision.

Dans le cadre du problème SAT, l'utilisation de la recherche locale, en prétraitement ou pendant la recherche, a déjà montré son efficacité. Par exemple, Mazure *et al.* [19] ont montré de manière expérimentale qu'effectuer une recherche locale en prétraitement permettait d'obtenir des informations intéressantes concernant les parties difficiles du problème (en particulier les clauses appartenant au noyau minimalement inconsistant). Plus récemment, Audemard *et al.* [1] ont utilisé le concept de *boundary* (proche du concept de variables FAC) et ont montré que l'approche résultante améliorait grandement les performances d'un solveur de type CDCL. L'utilisation du concept de variables FAC et de la recherche locale dans le cadre d'une approche hybride semble donc être très prometteuse. L'approche FAC-SOLVER décrite dans la section suivante tente d'implanter et de valider cela de manière expérimentale en prenant en compte un large panel d'instances.

4 L'approche FAC-SOLVER

Le solveur FAC-SOLVER intègre trois types d'approches en son sein : une recherche locale, un solveur de type MAC et un solveur hybride qui est une combinaison d'un solveur de recherche locale et du processus de filtrage GAC. Ces composantes interagissent ensemble de plusieurs manières (pondération, détection de FAC...) et partagent l'ensemble des informations obtenues au cours de recherche (variables FAC, valeurs supprimées au niveau 0). L'automate décrit dans la Figure 1 donne le schéma général de notre approche. Pour commencer, le processus appelle la recherche locale avec le problème initial. Dans le cas où la recherche locale échoue à trouver une solution dans le temps qui lui a été imparti (contrôlé par $slsProgress$), la partie hybride (RL+GAC) prend la main. Cette partie consiste à rendre taboues les variables problématiques pour la recherche locale en les fixant à l'aide d'un processus complet (affectation et filtrage). Enfin, notre approche passe la main à la partie MAC lorsqu'elle n'a pas été capable de fournir un résultat avant d'avoir atteint un nombre de conflits fixé à l'avance. Cette dernière partie est simplement un solveur MAC classique avec une heuristique de choix de variables basée sur la notion de variable FAC présentée précédem-

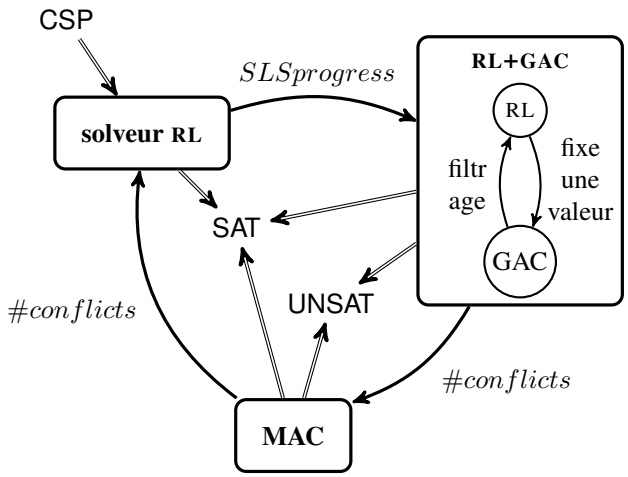


FIGURE 1 – Interactions entre les différentes composantes de FAC-SOLVER

ment. Comme pour la partie RL+GAC, la partie MAC redonne finalement la main à la recherche locale (nouveau cycle) lorsqu'un certain nombre de conflits a été atteint.

4.1 FAC-SOLVER

L'algorithme 1 décrit le solveur FAC-SOLVER. En premier lieu, l'ensemble des variables utiles aux différentes composantes de notre solveur est initialisé : le nombre de conflits contrôlant le redémarrage associé aux parties RL+GAC et MAC du solveur est initialisé à 10, l'ensemble contenant les variables FAC detectées est affecté à vide et une interprétation complète des variables est générée de manière aléatoire (ligne 2-6). Cette interprétation sera utilisée par la recherche locale. Ensuite, un appel à la procédure GAC est effectué afin d'assurer l'arc-consistance ou de montrer que le problème est directement incohérent (lignes 4-5). Une fois ce préambule terminé, tant qu'une solution n'a pas été trouvée ou que l'absence de solution n'a pas été établie, le solveur effectue de manière séquentielle les trois composantes RL, RL+GAC et MAC.

Avant de détailler chacune des composantes, nous décrivons, de manière intuitive, leurs interactions et apports respectifs. Pour la RL, en plus d'essayer de trouver une interprétation satisfaisant l'ensemble des contraintes du CSP, cette composante permet de détecter et collecter des variables FAC. Concernant la partie RL+GAC, elle permet d'aider la recherche locale (en vérifiant que certaines de ces affectations sont bien arc-cohérentes) et permet pondérer les contraintes les plus souvent falsifiées durant la RL (choix des variables limité au *scope* des contraintes falsifiées). Quant à la composante MAC, elle tente de résoudre le problème en intégrant l'ensemble des informations recueillies par les deux autres composantes (FAC et ajustement du poids des contraintes).

Algorithm 1: FAC-solver

Data: Un CSP $\mathcal{CN} = \langle \mathcal{X}, \mathcal{C} \rangle$
Result: $true$ si le CSP est satisfiable, $false$ sinon

1 $result \longleftarrow unknown$;
2 $maxConf \longleftarrow 10$;
3 $\mathcal{S}_e \longleftarrow \emptyset$; `//ensemble de var FAC`
4 $\mathcal{A} \longleftarrow$ un assignement de $\mathcal{X}$ choisi aléatoirement ;
5 `GAC()` ;
6 **if** $\exists X \in \mathcal{X}$ *s.t.* $dom(X) = \emptyset$ **then return** $false$;
7 **while** *(*$result = unkown$*)* **do**
8 initialiser la variable $slsProgress$;
9 `RL(`$\langle \mathcal{X}, \mathcal{C} \rangle$`)` ;
10 **if** *(*$result \neq unkown$*)* **then return** $true$;
11 `Hybrid(`$\langle \mathcal{X}, \mathcal{C} \rangle$`)` ;
12 **if** *(*$result \neq unkown$*)* **then return** $result$;
13 `MAC(`$\langle \mathcal{X}, \mathcal{C} \rangle$`)` ;
14 **if** *(*$result \neq unkown$*)* **then return** $result$;
15 $maxConf \longleftarrow maxConf \times 1.5$;
16 `Backjump(`*0*`)`; `//redémarrage/nouveau cycle`

Il important de noter que, puisque $maxConf$ est augmenté de manière géométrique (ligne 16), la composante MAC donnera un résultat lorsque la borne ($maxConf$) deviendra plus grande que le nombre de conflits nécessaires pour résoudre le CSP. Ainsi, la complétude de notre méthode est garantie.

4.2 La composante recherche locale : RL

La procédure 2 présente de manière succincte la composante représentant la partie recherche locale de notre solveur. Cette procédure est basée sur l'approche *walksat* [24] utilisée habituellement dans le cadre de SAT. Elle utilise l'heuristique *novelty* comme critère d'échappement [20] aux minima locaux. Cette méthode identifie également des variables FAC à chaque minimum local. La variable contrôlant l'avancement de la recherche est $slsProgress$, elle est augmentée dans deux situations : quand une nouvelle valeur de maxCSP (nombre de contraintes falsifiées minimum jamais trouvé) est atteinte (ligne 13) et lorsqu'une nouvelle variable FAC est découverte (ligne 4). Elle est décrémentée dans le cas où un minimum local est atteint et qu'aucune variable FAC n'a été trouvée (ligne 6). Cette variable est initialisée à 10000 si le CSP est binaire et 1000 sinon. Cette manière d'estimer le progrès de la RL est inspirée des travaux introduit par Hoos *et al.* [12] concernant le réglage des différents paramètres utilisés en RL dans le cadre de SAT. Ce paramètre est fondamental pour l'efficacité de la l'hybridation et permet d'évaluer dynamiquement la progression de la méthode de recherche locale dans son exploration stochastique de l'espace de recherche. Lorsque la recherche locale échoue à trouver une solution du CSP et qu'elle se trouve « engluée », le test $slsProgress < 0$ (ligne 7) permet de passer la main à

l'algorithme hybride qui va utiliser cette situation d'échec comme point de départ.

Procedure 2 : `RL(`$\langle\mathcal{X},\mathcal{C}\rangle$`)`

1 **while** $\exists C \in \mathcal{C}$ *telle que* C *est falsifiée par* $\mathcal{A}$ **do**
2 **if** *minimum local est atteint* **then**
3 **if** $\exists$ *variable* FAC **then**
4 Ajouter une nouvelle variable FAC à $\mathcal{S}_e$;
5 $slsProgress \longleftarrow slsProgress + 1000$;
6 **else** $slsProgress \longleftarrow slsProgress - 1$;
7 **if** $slsProgress < 0$ **then return** ;
8 **else**
9 Changer la valeur dans $\mathcal{A}$ d'une variable de $\mathcal{X}$ en fonction du critère d'échappement novelty;
10 **else**
11 Changer la valeur dans $\mathcal{A}$ d'une variable de $\mathcal{X}$ tel que le nombre de contraintes falsifiées diminue;
12 **if** *un nouveau* $maxCSP$ *est obtenu* **then**
13 $slsProgress \longleftarrow slsProgress + 1000$;
14 $result \longleftarrow true$;

4.3 La composante hybrid : RL+GAC

La procédure 3 décrit la partie RL+GAC de notre solveur. Étant donnée la dernière interprétation $\mathcal{A}$ explorée par RL, cette interprétation ne permettait plus à la RL de progresser (suivant notre critère) et va donc être utilisée par la composante RL+GAC pour fixer une valeur à une variable. Tant qu'un certain nombre de conflits n'a pas été atteint (ligne 3), la procédure sélectionne une variable X appartenant à une contrainte falsifiée par $\mathcal{A}$ (en utilisant l'heuristique $dom/wdeg$ [3]) et tente de fixer une de ses valeurs à l'aide de la procédure FIX (ligne 4-6). La valeur est choisie de telle sorte que ce soit la valeur selon laquelle X est assigné dans $\mathcal{A}$. Une fois cette variable fixée le problème peut être montré incohérent ($result = false$) et la procédure se termine. Sinon, un appel à la procédure RL est effectué. Mais les variables fixées durant cette étape ne pourront pas être modifiées par la recherche locale.

La procédure FIX, qui peut être vue comme une partie d'un solveur MAC, permet de gérer la partie affectation et propagation. Elle consiste à assigner la valeur v à la variable X (ligne 1) et à effectuer l'arc-cohérence généralisée sur le problème résultant (ligne 3). Ensuite, tant que le CSP est conflictuel (*i.e.* il existe une variable dont le domaine est vide) la procédure effectue un *backtrack* permettant de rétablir l'état de chaque variable à l'état $level - 1$, décrémente le niveau, réfute la valeur v_{level} de la variable X_{level} affectée au niveau $level$ et effectue de nouveau GAC (lignes 8-12). Un niveau correspond donc au nombre de valeurs

Procedure 3 : `Hybrid(`$\langle\mathcal{X},\mathcal{C}\rangle$`)`

1 $level \longleftarrow 0$;
2 $\#conf \longleftarrow 0$;
3 **while** $(\#conf < maxConf)$ **do**
4 $X \longleftarrow$ choisir une variable $X \in false(\mathcal{X},\mathcal{C},\mathcal{A})$ à l'aide de l'heuristique $dom/wdeg$;
5 $v \longleftarrow$ the value of X in $\mathcal{A}$;
6 `FIX(`$\langle\mathcal{X},\mathcal{C}\rangle$, X, v`)` ;
7 **if** $(result = false)$ **then return** ;
8 `RL(`$\langle\mathcal{X},\mathcal{C}\rangle$`)` ;

fixées heuristiquement par RL+GAC ou MAC.

Procedure 4 : `FIX(`$\langle\mathcal{X},\mathcal{C}\rangle$, X, v`)`

1 $\langle\mathcal{X},\mathcal{C}\rangle \longleftarrow \langle\mathcal{X},\mathcal{C}\rangle|_{X=v}$;
2 $level \longleftarrow level + 1$;
3 `GAC()` ;
4 **while** $\exists X' \in \mathcal{X}$ *s.t.* $dom(X') = \emptyset$ **do**
5 **if** $level = 0$ **then**
6 $result \longleftarrow false$;
7 **return** ;
8 $\langle\mathcal{X},\mathcal{C}\rangle \longleftarrow$ `Backtrack()` ;
9 $level \longleftarrow level - 1$;
10 $\#conf \longleftarrow \#conf + 1$;
11 $\langle\mathcal{X},\mathcal{C}\rangle \longleftarrow \langle\mathcal{X},\mathcal{C}\rangle|_{X_{level} \neq v_{level}}$;
12 `GAC()` ;

4.4 La composante MAC

La procédure suivante décrit la procédure MAC.

La composante MAC commence avec le CSP initial modulo les valeurs filtrées au niveau 0 durant les appels précédents à la procédure `FIX` faits durant RL+GAC. Cette composante diffère des solveurs MAC classiques par l'heuristique de choix de variables. En effet, tant qu'un conflit n'a pas été atteint le prochain point de choix est choisi parmi l'ensemble des variables FAC. Ensuite, l'heuristique de choix de variables devient $dom/wdeg$ (line 8-9) afin de disperser la recherche sur différentes zones d'incohérence. Cet algorithme n'effectue pas nécessairement une recherche complète puisque si le nombre de conflits $\#conf$ devient supérieur à $\#conf$ avant d'avoir résolu le problème, alors le processus s'arrête et un nouveau cycle est réalisé (RL $\rightarrow$ RL+GAC $\rightarrow$ MAC $\hookleftarrow$). Afin de garantir la complétude de la méthode, le nombre de conflits autorisés est augmenté avant chaque nouveau cycle (ligne 15 de l'algorithme 1).

		NOVELTY			RL+GAC			MAC			FAC-SOLVER		
		SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT	SAT	UNS	TOT
2-EXT	ACAD	7	0	7	7	2	9	7	2	9	7	2	9
	PATT	106	0	106	100	38	138	83	38	121	99	39	138
	QRND	24	0	24	24	51	75	24	51	75	24	51	75
	RAND	206	0	206	197	105	302	194	110	304	193	106	299
	REAL	6	0	6	7	0	7	7	0	7	7	0	7
	TOTAL	349	0	349	335	196	531	315	201	516	330	198	528
2-INT	ACAD	38	7	45	37	40	77	37	40	77	38	40	78
	BOOL	0	1	1	0	1	1	0	1	1	0	1	1
	PATT	112	0	112	150	60	210	146	62	208	152	62	214
	REAL	47	74	121	74	102	176	75	103	178	75	103	178
	TOTAL	197	82	279	261	203	464	258	206	464	265	206	471
N-EXT	BOOL	70	1	71	74	75	149	74	70	144	74	74	148
	PATT	6	0	6	30	0	30	29	0	29	30	0	30
	QRND	43	0	43	40	40	80	33	40	73	45	40	85
	RAND	70	0	70	68	32	100	72	34	106	70	34	104
	REAL	41	29	70	45	114	159	47	115	162	47	115	162
	TOTAL	230	30	260	257	261	518	255	259	514	266	263	529
N-INT	ACAD	40	0	40	39	23	62	36	23	59	40	23	63
	BOOL	145	1	146	156	12	168	146	12	158	162	13	175
	PATT	88	5	93	103	19	122	95	20	115	102	18	120
	REAL	85	2	87	152	3	155	150	3	153	152	3	155
	TOTAL	358	8	366	450	57	507	427	58	485	456	57	513
	TOTAL	**1134**	**113**	**1247**	**1293**	**717**	**2010**	**1255**	**724**	**1979**	**1317**	**724**	**2041**

TABLE 1 – Résulats expérimentaux.

Procedure MAC ($\langle \mathcal{X}, \mathcal{C} \rangle$)

```
Backjump(0) ;                    // redémarrage
level ⟵ 0;
#conf ⟵ 0 ;
while (#conf < maxConf) do
    if X = ∅ then
        result ⟵ true ;
        return ;
    if (#conf = 0) and (∃X ∈ S_e ∩ X) then
        X ⟵ choisi une variable dans S_e ;
    else
        X ⟵ choisi une variable selon dom/wdeg ;
    v ⟵ choisi aléatoirement une valeur dans
    dom(X) ;
    FIX(⟨X, C⟩, X, v) ;
    if (result = false) then return ;
```

5 Résultats expérimentaux

Afin de valider de manière expérimentale notre approche, nous avons considéré l'ensemble des instances des deux dernières compétitions CSP [5, 6]. Ces instances sont constituées d'instances binaires et n-aires, aléatoires et industrielles, satisfiables et insatisfiables. Ces instances ont été divisées en quatre catégories : 635 instances codant des contraintes binaires en extension (2-EXT), 696 instances codant des contraintes binaires en intention (2-INT), 704 instances codant des contraintes n-aires en extension (N-EXT) et 716 instances codant des contraintes n-aires en intention (N-INT). Afin d'étudier séparément chacune des composantes, nous avons testé quatre méthodes : Walksat+Novelty qui est notre implémentation de la RL, RL+GAC, MAC et notre approche FAC-SOLVER. L'ensemble des tests a été conduit sur un Intel Xeon 3.2 GHz (2 G RAM) sous Linux 2.6. Nous avons limité le temps à 1200 secondes et la mémoire à 900 Mbytes.

Le tableau 1 récapitule les résultats, en terme de nombres d'instances SAT et UNSAT résolues par les différentes approches. Pour chaque catégorie d'instances le total est reporté. Sur ces lignes de totaux, les résultats du meilleur solveur sont grisés. Le principal constat que l'on peut tirer de ce tableau est que notre approche FAC-SOLVER résout globalement le plus d'instances (que ce soit SAT ou UNSAT), et qu'elle est la meilleure sur trois des quatre catégories d'instances considérées. Pour le dernier type d'instances (binaires en extension), nous pouvons voir que les meilleurs solveurs sont différents pour chaque catégorie (SAT, UNS(AT) et TOT(AL)) et que le nombre d'instances résolues par FAC-SOLVER est toujours très proche

du meilleur résultat obtenu.

La Figure 2, représentant cinq nuages de point, permet de comparer plus finement les résultats obtenus par FAC-SOLVER comparativement aux autres approches. Pour chaque couple possible, le nuage de points résultant permet de corréler le temps mis par chacune des deux méthodes pour résoudre une instance donnée. Pour toutes les figures, le solveur FAC-SOLVER est représenté sur l'axe des abscisses tandis que l'approche comparée est reportée sur l'axe des ordonnées. Les résultats, reportés en secondes, sont visualisés à l'aide d'une échelle logarithmique. Ces comparaisons ont été réalisées sur les instances SAT et UNSAT à l'exception de la comparaison avec Walksat+Novelty où seules les instances SAT ont été considérées, la méthode de recherche locale étant incapable de prouver incohérence d'un CSP. Les principales informations pouvant être extraites sont les suivantes :

- Il y a plus d'instances situées sur la ligne Y=1200 que sur la ligne X=1200. Ceci montre, comme cela est reporté dans le tableau 1, que FAC-SOLVER résout plus d'instances que les autres méthodes ;
- À l'exception du solveur MAC sur les instances UNSAT, on peut voir qu'il y a beaucoup de points au dessus de la diagonale, montrant ainsi que FAC-SOLVER est généralement plus efficace que les autres méthodes. De plus, pour les instances UNSAT, on peut voir que la différence entre les différentes paires de solveurs est plus faible que sur les instances SAT (les points sont moins dispersés et plus proches de la diagonale). En ce qui concerne la comparaison de notre approche avec RL+GAC, on peut voir que FAC-SOLVER est globalement plus efficace. Comparativement à l'approche MAC, notre méthode est meilleure sur les instances SAT (apport de la recherche locale) et légèrement moins bonne (en temps) sur les instances UNSAT. Ce décalage de temps sur les instances UNSAT s'explique par le temps utilisé par la recherche locale pour collecter des informations.

Une autre manière de valider la robustesse de notre approche est d'étudier l'impact de l'interprétation initiale choisie pour la partie RL. Pour cela, nous avons sélectionné 96 instances (de manière aléatoire) parmi l'ensemble des instances et pour chacune d'entre elles nous avons lancé notre approche 50 fois avec des interprétations initiales différentes. Les résultats montrent que le choix de l'interprétation ne modifie pas outre mesure les résultats obtenus par notre approche. En effet, lorsqu'une instance a été résolue au moins une fois, elle l'a été dans les 49 autres lancements dans 97 % des cas avec un écart moyen de 2.52 secondes.

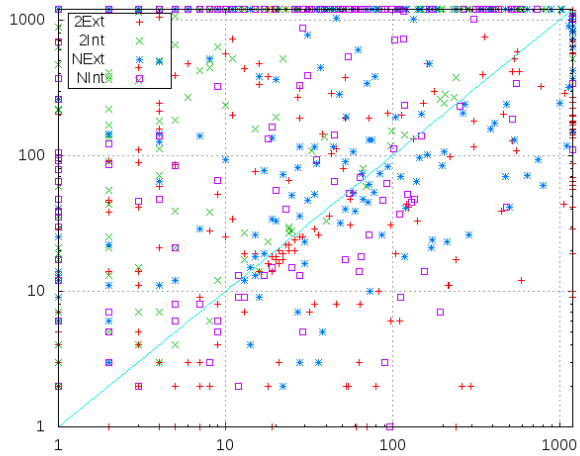
6 Perspectives et Conclusions

Dans ce papier, le concept de variables FAC a été introduit et étudié dans le cadre de la résolution du problème CSP. Le but de cette étude a été de développer un solveur hybride tirant parti de manière efficace des différentes approches présentes en son sein. Les résultats obtenus sur un très large panel d'instances ont permis de montrer que le but que nous nous étions fixé à été atteint.

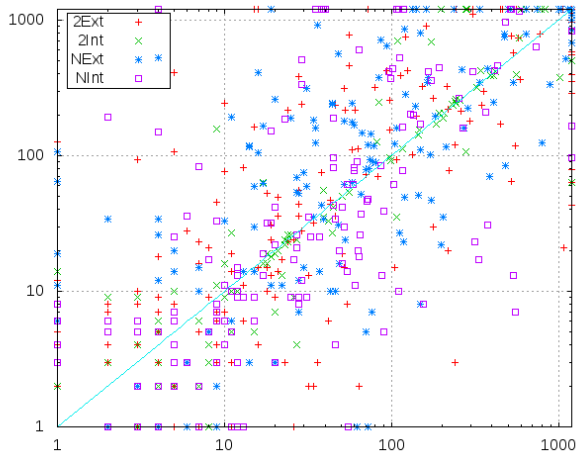
Une question se pose naturellement : dans quelle mesure chacune des composantes de notre approche prend-elle part à l'amélioration de l'efficacité du solveur ? D'après les expérimentations menées, chacune des composantes permettait de trouver la solution et chacune était nécessaires (variables FAC, RL, méthode hybride impliquant la RL et un processus de filtrage) pour assurer la domination de notre méthode. En particulier, nous avons mesuré que dans 56 % des instances considérées, des variables FAC ont été détectées et ont ainsi permis de jouer un rôle prépondérant dans la résolution du problème (même lorsque celui-ci est satisfiable).

L'algorithme FAC-SOLVER proposé est élémentaire et peut être amélioré par un réglage de ses différents paramètres de plusieurs manières. En particulier, une étude expérimentale plus poussée pourrait permettre d'optimiser les différentes variables de contrôle et les différents facteur d'augmentation, lesquels ont été fixés arbitrairement. De plus, notre implémentation n'inclut pas certaines des techniques de simplification utilisées habituellement dans le cadre de CSP, comme par exemple l'exploitation des symétries ou des contraintes globales. Nous pensons que l'intégration de ces techniques au sein de notre solveur permettrait d'augmenter de manière significative les performances. Il pourrait aussi être intéressant d'explorer une relaxation du concept de variables FAC en prenant en compte (en plus des variables FAC) les variables apparaissant le plus souvent dans les contraintes falsifiées. Cette approche pourrait être utile dans le cas où le CSP considéré ne possède pas de variable FAC.

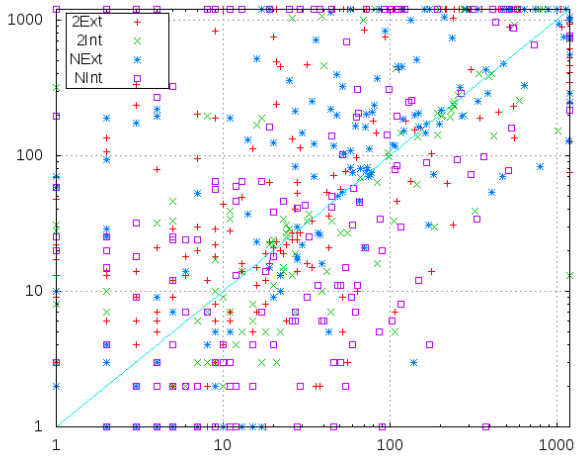
Finalement, nous pensons que le concept de variables FAC est un bon compromis entre : d'une part le faible coup de calcul nécessaire à la RL pour les détecter, et d'autre part, l'apport théorique concernant l'heuristique de choix de variable d'un solveur MAC afin d'obtenir une preuve de petite taille. Prenant part à l'ensemble des MUC du problème, les variables FAC permettent de se focaliser sur la partie difficile du problème. Cependant, il est facile de trouver des instances insatisfiables où les variables FAC ne prennent conceptuellement pas part à la cause réelle de l'insatisfiabilité, mais apparaissent simplement dans l'ensemble des variables de tous les MUC du problème (alors qu'elles ne fournissent pas véritablement d'information sur la cause du conflit). Raffiner le concept de variable FAC afin de capturer plus finement l'essence de l'insatisfiabilité tout en gardant une heuristique efficace (temps de calcul) constitue un vértiable challenge.



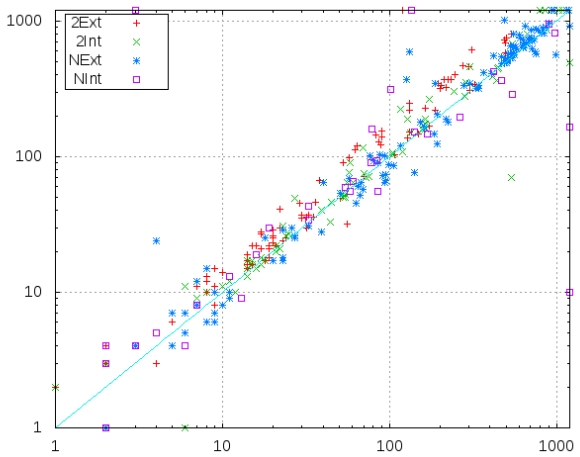
(a) FAC-SOLVER vs. NOVELTY



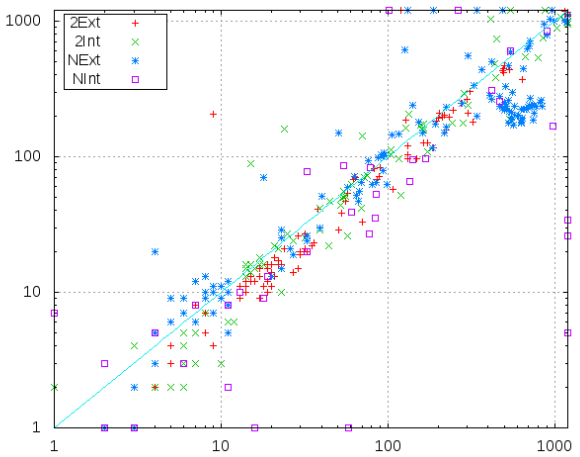
(b) FAC-SOLVER vs. RL+GAC



(c) FAC-SOLVER vs. MAC



(d) FAC-SOLVER vs. RL+GAC



(e) FAC-SOLVER vs. MAC

FIGURE 2 – Comparaison entre FAC-SOLVER et les autres approches : (a) (b) (c) instances SAT/(d) (e) instances UNSAT.

Références

[1] G. Audemard, J.-M. Lagniez, B. Mazure, and L. Saïs. Boosting local search thanks to cdcl. In *Seventeenth International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'10)*, pages 474–488, 2010.

[2] C. Bessière, J.-C. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2) :165–185, 2005.

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Saïs. Boosting systematic search by weighting constraints. In *Eleventh European Conference on Artificial Intelligence (ECAI'04)*, pages 146–150, 2004.

[4] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91)*, pages 331–340, 1991.

[5] Third international csp solver competition, 2008. `http://cpai.ucc.ie/08/`.

[6] Fourth international constraint solver competition, 2009. `http://cpai.ucc.ie/09/`.

[7] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence*, 57 :227–270, 1992.

[8] P. Galinier and J.-K. Hao. Tabu search for maximal constraint satisfaction problems. In *Third International Conference on Principles and Practice of Constraint Programming (CP'97)*, pages 196–208, 1997.

[9] E. Goldberg. Boundary points and resolution. In *SAT*, pages 147–160, 2009.

[10] J. Gu. Design efficient local search algorithms. In *Fifth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE'92)*, pages 651–654, 1992.

[11] Holger H. Hoos. An adaptive noise mechanism for walksat. In *AAAI/IAAI*, pages 655–660, 2002.

[12] E. Hébrard. Mistral 1.529, 2006. `http://4c.ucc.ie/~ehebrard/mistral/doxygen/html/main.html`.

[13] N. Jussien and O. Lhomme. Combining constraint programming and local search to design new powerful heuristics. In *Fifth Metaheuristics International Conference (MIC'2003)*, 2003.

[14] C. Lecoutre and F. Hemery. A study of residual supports in arc consistency. In *Twentieth International Joint Conference on Artificial Intelligence (IJCAI'2007)*, pages 125–130, 2007.

[15] C. Lecoutre and S. Tabary. Abscon 112 : towards more robustness. In *Third International Constraint Solver Competition (CSC'08)*, pages 41–48, 2008.

[16] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1) :99–118, 1977.

[17] B. Mazure, L. Saïs, and E. Grégoire. Boosting complete techniques thanks to local search methods. *Annals of Mathematics and Artificial Intelligence*, 22(3-4) :319–331, 1998.

[18] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 321–326, 1997.

[19] S. Merchez, C. Lecoutre, and F. Boussemart. Abscon : A prototype to solve csps with abstraction. In *Seventh International Conference on Principles and Practice of Constraint Programming (CP'01)*, pages 730–744, 2001.

[20] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing conflicts : A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3) :161–205, 1992.

[21] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Eleventh European Conference on Artificial Intelligence (ECAI'94)*, pages 125–129, 1994.

[22] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, 1994.

[23] B. Selman, H.J. Levesque, and D.G. Mitchell. A new method for solving hard satisfiability problems. In *Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.

[24] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear csp into sat. *Constraints*, 14(2) :254–272, 2009.

[25] Choco Team. Choco : an open source java constraint programming library. Research report 10-02-INFO, Ecole des Mines de Nantes, 2010.