De FC à MAC : un algorithme paramétrable pour la résolution des CSP

Assef Chmeiss, Lakhdar Sais

De FC à MAC : un algorithme paramétrable pour la résolution des CSP

Assef Chmeiss **Lakhdar Saïs**

CRIL CNRS – Université d'Artois
rue Jean Souvraz - SP 18
62307 Lens Cedex
{chmeiss, sais}@cril.univ-artois.fr

Résumé

Beaucoup d'algorithmes de résolution de Problèmes de Satisfaction de Contraintes ont été proposés ces dernières années. Parmi ces algorithmes nous pouvons mentionner les deux les plus populaires et les plus étudiés : le Forward-Checking (FC) et Maintaining Arc-Consistency (MAC). Dans ce papier, nous étudions ces deux algorithmes et nous réévaluons leurs performances sur des problèmes générés aléatoirement. Précisément, nous montrons expérimentalement que FC est meilleur que MAC sur des CSP difficiles dont le graphe de contraintes est très dense et la dureté des contraintes est faible. En revanche, MAC se montre plus performant que FC sur des problèmes difficiles avec un graphe de contraintes peu dense et une dureté élevée. Ces résultats montrent que le maintien de l'Arc-consistance pendant la recherche peut être une perte de temps. Ensuite, Nous proposons une approche générique qui permet, pendant la recherche, un maintien partiel et paramétrable de la consistance locale.

Abstract

Many backtrack search algorithms has been designed over the last years to solve constraint satisfaction problems. Among them, Forward Checking (FC) and Maintaining Arc Consistency (MAC) algorithms are the most popular and studied algorithms. In this paper, such algorithms are revisited and extensively compared giving rise to interesting characterization of their efficiency with respect to random instances. More precisely, we provide experimental evidence that FC outperforms MAC on hard CSPs with high graph density and low constraint tightness whereas MAC is better on hard CSPs with low density and high constraints tightness. This results show that on some CSPs maintaining full arc consistency during search might be time consuming. Then, we propose a new generic approach that maintain partial and parameterizable form of local consistency.

1 Introduction

Les Problèmes de Satisfaction de Contraintes (CSP) sont largement utilisés dans la résolution de problèmes combinatoires apparaissant dans diverses applications. Ils concernent la recherche de solutions dans un *réseaux de contraintes* c'est-à-dire affecter des valeurs à des variables,tout en satisfaisant un ensemble de contraintes portant sur ces différentes variables. La technique classique utilisée pour résoudre un CSP est le backtrack systématique. Cette procédure (backtrack) choisit, de façon répétitive, une variable et essaye de lui affecter l'une des valeurs dans son domaine. Ensuite, soit elle choisit une nouvelle variable soit elle effectue un retour en arrière en cas d'échec. Cette technique est à la base de la plupart des algorithmes de résolution des CSP. Néanmoins, pour résoudre certains problèmes combinatoires difficiles, il faut améliorer la performance de cette procédure à l'aide de techniques plus intelligentes. Une amélioration importante à ajouter est le filtrage, qui consiste à enlever des domaines futurs des valeurs incompatibles avec l'instanciation courante. Beaucoup de travaux ont étudié différents niveaux de filtrage qui peut être appliqué à chaque nœud de l'arbre de recherche. Les deux algorithmes les plus connus dans ce cadre sont FC (Forward-Checking) et MAC (Maintaining Arc-Consistency). La performance de ces algorithmes a été discutée dans plusieurs travaux dans la littérature [1, 5, 16, 17]. Il semble que ces travaux soient d'accord que MAC constitue un bon choix pour la résolution des CSP [5, 17, 20]. Par conséquent, beaucoup d'améliorations de la complexité temporelle et/ou spatiale des algorithmes de consistance d'arc ont été proposées[3, 4, 7, 15, 21].

Nous signalons que les conclusions concernant la

performance des algorithmes de résolution de CSP sont tirées à partir de résultats expérimentaux. La complexité de ces algorithmes reste dans le pire des cas de l'ordre de d^n avec d la taille des domaines et n le nombre de variables. Les expérimentations sont, en général, effectuées sur des instances générées aléatoirement et sur un nombre limité de problèmes réels.

Nous pensons que ces deux algorithmes (FC et MAC) sont complémentaires et que leur performance dépend des problèmes testés. En d'autres termes, la rentabilité du maintien de la consistance d'arc pendant la recherche dépend des caractéristiques des CSP testés.

Dans ce papier, nous présentons une réévaluation comparative de FC et MAC sur des problèmes aléatoires avec des caractéristiques différentes comme la densité du graphe de contraintes et la dureté des contraintes. Plus précisément, nous fournissons des résultats montrant que FC est meilleur que MAC sur des CSP difficiles dont le graphe de contraintes est très dense et avec une faible dureté. En s'appuyant sur ces observations, nous proposons un nouvel algorithme générique de backtrack appelé $MAC(dist_k)$. Cet algorithme maintient, pendant la recherche, une forme partielle de consistance d'arc par rapport à un paramètre k appelé *distance*. Nous montrons, sur quelques instances, qu'il existe une distance i pour laquelle $MAC(dist_i)$ améliore à la fois FC (qui filtre uniquement le voisinage de la variable) et MAC (qui maintient la consistance d'arc).
Ce article organisé comme suit. Après un rappel de quelques définitions, nous rappelons les algorithmes FC et MAC. Ensuite, nous présentons le nouvel algorithme générique $MAC(dist_k)$ qui permet de paramétrer le niveau de consistance d'arc à maintenir. Nous montrons, ensuite, une comparaison expérimentale entre FC et MAC sur des problèmes aléatoires difficiles. Nous présentons également des résultats préliminaires sur le comportement de $MAC(dist_k)$ avec plusieurs valeurs de k. Enfin, nous donnons quelques directions possibles sur des futurs travaux et nous terminons par une conclusion sur ce papier.

2 Préliminaires

Un CSP (Problème de Satisfaction de Contraintes) à domaines finis est défini par le triplet $\mathcal{P} = (X, D, C)$:

- $X = \{x_1, x_2, \dots x_n\}$ est un ensemble de n variables.
- $D = \{d_1, d_2, \dots d_n\}$ est un ensemble de domaines finis, une variable x_i prend ses valeurs dans son domaine d_i. $|d_i|$ représente la taille du domaine d_i. Si une variable x_i possède une seul valeur dans son domaine ($|d_i| = 1$), elle est appelée singleton valeur.
- $C = \{c_1, c_2, \dots, c_m\}$ est un ensemble de m contraintes. Chaque contrainte c_i est une paire $(vars(c_i), rel(c_i))$ telle que :
  - $vars(c_i) \subset X$ est un sous-ensemble de variables. $|vars(c_i)|$ représente l'arité de la contrainte.
  - $rel(c_i) \subset \prod_{x_k \in vars(c_i)} d_k$ est la relation définie par la contrainte c_i. Elle spécifie les couples de valeurs autorisés par la contrainte. Nous appelons *test de consistance* le test d'appartenance d'un couple de valeurs à la relation $rel(c_i)$.

Une *solution* est une affectation des valeurs aux variables qui satisfait toutes les contraintes. Étant donné un CSP, les questions qui se posent sont de savoir s'il existe une solution, d'en exhiber une si tel est le cas, ou encore de trouver toutes les solutions. La première question permet de définir un problème NP-complet. Pour des raisons de simplicité nous nous limitons, dans ce papier, aux réseaux de contraintes binaires, où les contraintes n'impliquent que deux variables. Dans le cas des CSP binaires, une contrainte entre les variables x_i et x_j est notée C_{ij}. Nous pouvons associer à tout CSP binaire $\mathcal{P} = (X, D, C)$ un graphe $\mathcal{G}_\mathcal{P} = (X, C)$ appelé *le graphe de contraintes* dans lequel les nœuds sont les variables du réseau, et une arête relie une paire de nœuds si et seulement si il y a une contraintes sur les variables correspondantes. L'ensemble des nœuds qui partagent une arête avec x_i (le voisinage de x_i) est notée $\Gamma(x_i)$. Ainsi, $|\Gamma(x_i)|$ représente le degré de x_i. Une variable x_i dont le degré est égal à 1 sera appelé *singleton degré*. Un domaine d_i est arc-consistant ssi, $\forall a \in d_i, \forall x_j \in X$ tel que $c_{ij} \in C$, $\exists b \in d_j$ t.q. $(a, b) \in rel(c_{ij})$. b est appelé *support* de a par rapport à la contrainte c_{ij}. Un CSP est arc-consistant ssi $\forall d_i \in D, d_i \neq \emptyset$ et d_i est arc-consistant.

3 Techniques de résolution des CSP

Pour résoudre un CSP, on peut employer un algorithme de recherche en profondeur de type backtracking (BT) [6, 13]. A chaque étape de l'arbre de recherche, on essaye d'affecter une variable, si cette affectation produit un conflit un retour en arrière est effectué. Dans l'objectif de réduire l'espace de recherche, beaucoup d'améliorations concernant l'algorithme BT ont été proposées. Ces améliorations concernent essentiellement les questions suivantes : quel type de consistance locale doit-on appliquer ? Quelle est la prochaine variable à affecter ? et que doit-on faire en cas de conflit ?

Les techniques qui ont été proposés dans l'objectif de prendre en compte les questions précédentes peuvent être classifier en deux catégories : les ap-

proches prospectives (look-ahead) et les approches rétrospectives (Look-back) :

- Les *approches prospectives* : elles détectent à l'avance des futures situations d'échec. Elles effectuent une sorte de filtrage dans les domaines des variables non encore instanciées. Elles suppriment donc les valeurs incompatibles avec l'instanciation (solution partielle) courante. Ce filtrage est, en général, réalisé à l'aide de techniques basées sur des propriétés de consistance locale. Les algorithmes les plus connus dans ce cadre sont le Forward Checking et MAC (Maintaining Arc Consistency).
- Les *approches rétrospectives* : elles consistent à déterminer la cause de l'échec et ensuite effectuer un retour en arrière directement vers la variable responsable de l'échec et non pas vers la variable précédente. On peut ainsi éviter des retours en arrières inutiles. Parmi ces techniques, nous pouvons citer l'algorithme de backjumping [11], conflict-directed backjumping [16] et dynamic backtracking [12].

D'autres approches combinent les deux précédentes. Par exemple, on peut intégrer conflict-directed backjumping avec forward-checking (FC-CBJ) [16]. Il semblerait qu'il y a une relation inverse entre les approches prospectives et les approches rétrospectives, plus on effectue du filtrage en avant moins on fait de retour en arrière. Pour plus de détails voir le livre de Rina Dechter [8].

La complexité en temps de ces algorithmes reste exponentielle dans le pire des cas. Par conséquent, la comparaison des performances de ces algorithmes est basée essentiellement sur l'analyse de résultats expérimentaux obtenus sur différentes classes de CSP. Ces résultats permettent d'observer le comportement de ces algorithmes. Il semble que MAC soit le meilleur procédure de résolution de CSP et particulièrement sur des CSP difficiles de grande taille. Par exemple, Bessière et Regin [5] ont affirmé : "MAC is not only more efficient than FC to solve large practical problems, but it is also really more efficient than FC on hard and large random problems".

D'autres conclusions concernant l'efficacité d'autres algorithmes de résolution se trouvent dans la littérature[1, 16]. Nous pensons que les performances des différentes algorithmes de recherche sont liées au choix fait :

- dans la façon dont ces algorithmes sont implémentés : heuristiques utilisées pour le choix de variables et/ou de valeurs, combinaison d'approches prospectives et rétrospectives.
- dans les caractéristiques des instances testées : taille, densité du graphe de contraintes, dureté des contraintes.

Notons que, dans la communauté SAT, les techniques prospectives sont en général très efficaces sur des instances aléatoires difficiles. D'un autre coté, les approches rétrospectives se montrent plus performantes sur des problèmes réels [19, 2, 22, 14, 9]. Notre intuition est que ces observations seraient valables dans les cadre des CSP.

Dans ce papier, nous nous intéressons au comportement des approches prospectives et leur performance sur des problèmes générés aléatoirement. Plus précisément, nous considérons les deux algorithmes les plus populaires : Forward-Checking (qui maintient une forme partielle de filtrage) et MAC (qui effectue un filtrage par consistance d'arc sur l'ensemble des variables non affectées. L'objectif principal de ce travail n'est pas de déterminer lequel des deux algorithmes domine l'autre. Nous cherchons à caractériser les instances sur lesquelles l'un est meilleur que l'autre. Notre intuition est que le niveau de filtrage (partiel ou total) à appliquer dépend des problèmes testés, plus la dureté des contraintes est élevée, plus le filtrage en avant est utile.

4 FC versus MAC

Dans cette section, nous comparons la performance de FC et MAC sur des problèmes aléatoires. Nous décrivons les principaux points concernant l'implémentation de ces deux algorithmes. Dans les deux cas, nous employons les mêmes heuristiques et la même étape de pré-traitement :

- *heuristiques* : la prochaine variable à affecter est choisie selon l'heuristique la plus efficace *dom/deg* [5]. Cette heuristique choisit la variable qui minimise le rapport entre la taille de domaine et le degré de la variable.
- *pré-traitement :* une étape de filtrage est réalisée par consistance d'arc avant la résolution. Nous utilisons l'algorithme AC8 [7] pour effectuer cette étape.

Tandis que FC réalise une forme limitée de filtrage pendant la recherche (propagation par rapport aux voisins futurs de la variable courante), MAC maintient à chaque étape de la recherche un CSP arc-consistant. Ainsi, MAC peut grâce au maintien de la consistance d'arc déconnecter les variables singleton valeur et singleton degré[18].

Les expérimentations ont été effectuées sur des CSP générés aléatoirement selon le modèle B [10]. Ce générateur admet quatre paramètres :

- le nombre de variables N
- la taille initiale des domaines D
- la proportion $p1$ des contraintes dans le réseau (ou le nombre de contraintes C donné par $C = p1 * N \cdot (N-1)/2$)

– la proportion $p2$ des couples de valeurs interdits par les contraintes (ou T le nombre de couples interdits $T = p2 * D * D$)

Dans la suite, nous utilisons C et T au lieu de $p1$ et $p2$. C est appelé la *densité* du graphe de contraintes et T la *dureté* des contraintes. Les classes testées seront donc désignées par le quadruplet $< N, D, T, C >$

Avant de présenter une comparaison expérimentale entre FC et MAC, nous rappelons quelques résultats extraits de l'article de Bessière et Regin [5]. Dans leur article, ils comparent la performance de FC-CBJ et MAC sur des CSP aléatoires. Les résultats montrent que MAC est plus performant que FC-CBJ qui est généralement meilleur que FC. On peut, donc, dire que MAC est plus efficace que FC. Le rapport d'efficacité donné (si on considère le temps d'exécution) varie entre 2.16 et 476 (MAC peut être 476 plus rapide que FC-CBJ). La table 1, rappelle les classes testées dans l'article mentionné [5]

$< N, D, T, C >$
$< 35, 6, 4, 501 >$
$< 35, 9, 27, 178 >$
$< 50, 6, 8, 325 >$
$< 50, 20, 300, 95 >$
$< 100, 12, 110, 120 >$
$< 125, 3, 1, 929 >$
$< 350, 3, 1, 2292 >$
$< 250, 3, 3, 391 >$
$< 350, 3, 3, 524 >$

TAB. 1 – Classes extraites de [5]

A l'exception des trois premières classes (elles ne sont pas vraiment difficiles), les autres classes correspondent à des problèmes dont le graphe de contraintes est peu dense. En plus, les problèmes issus des deux dernières classes sont arc-inconsistants ce qui peut être prouvé par une étape de pré-traitement par consistance d'arc.

Nous avons effectué des expérimentations sur des CSP aléatoires afin de comparer FC et MAC. Nous avons choisi des classes pour lesquelles MAC est meilleur que FC et d'autres pour lesquelles FC est plus performant. Les résultats sont reportés dans deux tables, la première montrent les problèmes où FC se montre plus efficace tandis que MAC est meilleur sur les classes de la deuxième table.

Les critères que nous considérons dans la comparaison sont : le nombre de tests de consistance (`#checks`), le nombre de nœuds visités(`#nodes`) et le temps d'exécution exprimé en secondes[1].

[1]Les expérimentations ont été réalisées sur un PC Pentium4 2.8 sous Linux

Nous avons réalisé des expérimentations sur plusieurs classes en modifiant le nombre de variables N et la taille des domaines D. Dans l'objectif de traiter des problèmes proches de la phase de transition, nous avons cherché pour chaque couple (N, D) la valeur appropriée pour la dureté (T) et le nombre de contraintes (C).

La table 2, présente les résultats sur une variété de classes. Pour chaque classe, nous avons généré 20 ou 10 CSP (la deuxième colonne précise le nombre de problèmes testés) et les résultats reportés correspondent à la moyenne des résultats sur les CSP testés. La première colonne présente la classe testée $(< N, D, T, C >)$. La deuxième indique le nombre de problèmes consistants et le nombre de problèmes inconsistants (une ligne par catégorie). La troisième et la quatrième colonnes donnent le nombre de tests de consistance effectués par FC et MAC respectivement. La cinquième et la sixième colonnes présentent le nombre de nœuds visités par FC et MAC. Enfin, les deux dernières colonnes fournissent le temps d'exécution de chaque algorithme. Nous signalons que nous avons choisi de présenter les résultats sur les problèmes consistants et sur ceux inconsistants séparément. Ce choix permet d'avoir une idée sur le comportement des algorithmes sur ces deux catégories.

La table 2, montre que FC est meilleur que MAC sur toutes les classes si on considère le nombre de tests de consistance et le temps d'exécution comme mesures d'efficacité. Nous pouvons remarquer que le graphe des contraintes des classes testées est dense (pour certaines classes, quand $N = 20$, le graphe est complet), ce qui fait la particularité de ces classes. En général, plus le graphe est dense, plus FC se montre performant. FC peut être 2 à 5 fois plus rapide que MAC selon la densité du graphe. Par exemple, le rapport d'efficacité sur les classes $< 30, 20, 177, 150 >$ et $< 60, 10, 20, 600 >$ (densité de graphe est de 33%) est d'environ 2. Ce rapport peut arriver à 5 sur les classes dont le graphe est complet (la classe $< 20, 20, 108, 190 >$). Ces résultats montrent que FC est plus performant que MAC sur des CSP dont le graphe de contrainte est dense.

Dans la table 3, nous présentons quelques résultats montrant que MAC est plus efficace que FC sur des CSP avec un graphe de contraintes peu dense. Nous avons testé la classe $< 90, 20, 272, 222 >$ dont la densité de son graphe est d'environ 5.5%. La table donne la moyenne des résultats sur 20 problèmes aléatoires. Nous remarquons que, sur cette classe, MAC est meilleur que FC, le rapport de performance entre les deux est de 6.

$< N, D, T, C >$		#checks (thousands)		#nodes		CPU time(sec.)	
		FC	MAC	FC	MAC	FC	MAC
$< 20, 20, 108, 190 >$	Sat (8/20)	4738	24710	69707	7127	0.30	1.54
	Unsat (12/20)	11507	60034	169689	17386	0.35	3.60
$< 20, 30, 271, 190 >$	Sat (11/20)	55019	297723	750857	71605	4.43	18.84
	Unsat (9/20)	102844	556846	1403053	133778	8.02	35.23
$< 20, 40, 515, 190 >$	Sat (17/20)	85617	469758	1107022	99627	7.51	31.92
	Unsat (3/20)	514785	2823836	6679140	601434	43.96	200.92
$< 30, 20, 177, 150 >$	Sat (10/20)	3455	12861	59267	3434	0.23	0.81
	Unsat (10/20)	5702	23611	89617	5695	0.68	1.26
$< 30, 30, 261, 300 >$	Sat (8/20)	1976781	11377886	23221965	2234677	160.15	768.95
	Unsat (12/20)	3152289	18351158	36873661	3579692	254.55	1239.55
$< 50, 10, 20, 500 >$	Sat (7/20)	70207	243056	1122769	93493	5.31	13.25
	Unsat (13/20)	181589	630282	2866452	238128	13.68	33.41
$< 60, 10, 20, 600 >$	Sat (3/10)	748914	2286254	11893358	850931	59.47	124.76
	Unsat (7/10)	1353173	4246448	21159717	1532982	108.97	230.96
$< 60, 10, 14, 900 >$	Sat (2/10)	3645901	14043816	42444545	4173429	278.93	806.02
	Unsat (8/10)	11112147	42403058	130670604	12916752	874.06	2451.51

TAB. 2 – Comparaison entre FC et MAC

Procedure		#checks(thousands)	#nodes	CPU time(sec.)
FC	Sat (12/20)	764051	13515952	56.21
	Unsat (8/20)	1393907	24848571	102.92
	All	1015993	18049000	74.89
MAC	Sat (12/20)	232860	41153	9.11
	Unsat (8/20)	459242	75253	17.74
	All	323412	54793	12.56

TAB. 3 – FC versus MAC : <90, 20, 272, 222>

Les résultats présentés dans cette sections fournissent une réponse définitive à la question : quel algorithmes est toujours meilleur, MAC ou FC ? En tout cas, nous avons confirmation que FC est plus performant que MAC sur des CSP difficiles avec un graphe de contraintes très dense et une dureté faible de contraintes. D'un autre coté, MAC est plus efficace sur des CPS difficiles avec un graphe peu dense et une dureté de contraintes élevée. Cette section donne une première idée sur les caractéristiques des problèmes sur lesquels FC est plus utile que MAC et vice-versa. Dans la section suivante, nous proposons un algorithme générique qui exploite une forme partielle de filtrage pendant la recherche.

5 L'algorithme $MAC(dist_k)$

Le comportement de FC et MAC sur des CSP difficiles dépend de deux caractéristiques importantes : la densité du graphe de contraintes et la dureté des contraintes. Les résultats présentés dans la section précédente montrent que le maintien total de la consistance d'arc à chaque étape de l'arbre de recherche peut s'avérer une perte de temps. Dans cette section, nous explorons des niveaux intermédiaires de consistance d'arc. Ainsi, nous proposons un nouvel algorithme générique de recherche appelé $MAC(dist_k)$. Cet algorithme maintient, pendant la recherche, une forme partielle de consistance d'arc par rapport à un paramètre de distance k. Le but est de montrer l'existence d'une valeur de k pour laquelle $MAC(dist_k)$ est meilleur que FC et MAC à la fois.

Nous donnons quelques définitions et notations nécessaires pour la suite de cet article.

Définition 1 *Soit $\mathcal{P} = (X, D, C)$ un CSP, $x_i, x_j \in X$ deux variables. Nous définissons $dist(x_i, x_j)$ comme étant le plus court chemin entre x_i et x_j dans le graphe associé $\mathcal{G}_\mathcal{P}$.*

Définition 2 *Soit $\mathcal{P} = (X, D, C)$ un CSP et $x_i \in X$. $\Gamma_{dist}(x_i, k) = \{x_j | dist(xi, x_j) \leq k\}$ désigne le voisinage de x_i à la distance k.*

Exemple 1 *Nous illustrons les définition précédentes par le graphe dans la figure 1.*
- *$dist(x, t) = 2$ et $dist(t, r) = 4$.*
- *$\Gamma_{dist}(x, 1) = \{u, w, y, z\}$.*

Définition 3 *Soit $\mathcal{P} = (X, D, C)$ un CSP. $P_k(x_i) = (X', D, C')$ est défini comme étant le CSP induit par $\Gamma_{dist}(x_i, k)$ t.q. $X' = \Gamma_{dist}(x_i, k)$ et $C' = \{C_{ij} | dist(x_i, x_j) <= k\}$.*

Définition 4 *Soit $\mathcal{P} = (X, D, C)$ un CSP et $x_i \in X$. $\mathcal{P}$ est k-arc-consistant par rapport à x_i ssi $P_k(x_i)$ est arc-consistant.*

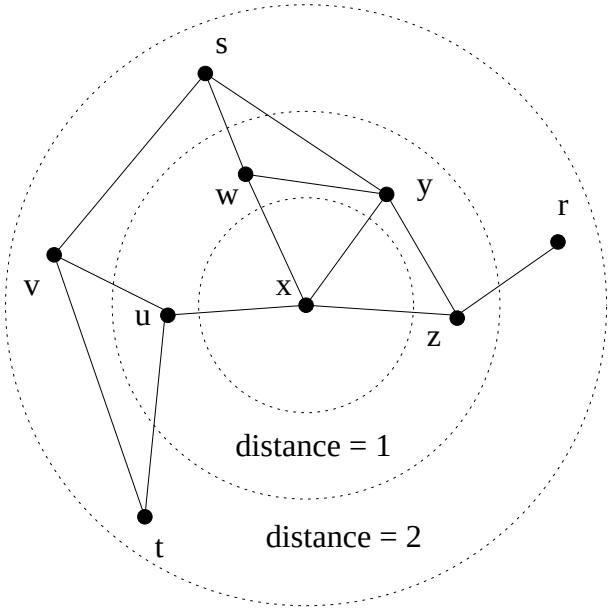

FIG. 1 – Graphe de contraintes et distances

Nous pouvons maintenant définir $MAC(dist_k)$ comme l'algorithme qui maintient la k-arc-consistance à chaque étape de l'arbre de recherche. Ainsi, $MAC(dist_0)$ correspond à l'algorithme de backtrack classique (BT), et $MAC(dist_{max})$, où max est la distance maximum entre les paires de variables dans le graphe de contraintes, correspond à l'algorithme MAC.

Nous signalons que $MAC(dist_1)$ ne correspond pas tout à fait à Forward-Checking. En effet, FC propage l'effet de l'affectation d'une variable x_i à toutes les variables futures x_j connectées à x_i dans le graphe. De son coté, $MAC(dist_1)$ réalise des propagations additionnelles sur toutes les contraintes se trouvant à une distance égale à 1 par rapport à x_i. Dans l'exemple de la figure 1, les contraintes C_{wy} et C_{yz} sont également révisées.
L'algorithme 1, donne une description de $Mac(dist_k)$. Dans l'étape d'initialisation, la distance (le plus court chemin) entre toute paire de variables est calculée. Nous réalisons également une étape de pré-traitement par consistance d'arc sur le CSP initial. Étant donnée une distance k, la fonction *propagate* restreint l'application de l'arc-consistance à $P_k(x_i)$.

Les deux opérations effectuées dans MAC (déconnexion des variables singleton degré et singleton valeur) peuvent être étendues à $MAC(dist_k)$. Quand on affecte une variable x_i, seulement les variables singleton degré et singleton valeur x_j qui vérifient la condition $dist(x_i, x_j) < k$ sont déconnectées.

Algorithm 1: Algorithm $MAC(dist_k)$
fonction $MAC(dist_k)$ **return** *boolean*
Input: $\mathcal{P}$: CSP, k : integer
Output: *consistent* : boolean (*true* if $\mathcal{P}$ is consistent, *false* otherwise)

```
begin
    initialization(k);
    consistent :=true, level :=1;
    while 1 ≤ level ≤ n do
        if consistent then
            x_level := chooseVariable();
            v_level :=chooseValue(x_level);
            instantiate(x_level,v_level);
            consistent := propagate(x_level, v_level, k);
        if consistent then
            level :=level+1;
        else
            level :=level-1;
    return level=n+1
end
```

L'une des étapes les plus importante dans $MAC(dist_k)$ est l'étape de propagation (l'appel à *propagate*). Nous décrivons brièvement la fonction de propagation. La fonction *propagate* établit un CSP arc-consistant sur $P_k(var)$ où var est la variable en cours d'affectation. Comme nous pouvons le remarquer, seulement les variables à une distance inférieure à (ou égale à) k sont considérées. La fonction *revise* assure que le domaine de la variable y vérifie la consistance d'arc.

Algorithm 2: propagation
fonction *propagate* **return** *boolean*
Input: var : integer, val : integer, k : integer
Output: *consistent* : boolean (*true* if $P_k(var)$ is Arc-Consistent, *false* otherwise)

```
begin
    currentDomain_var = {val};
    push(var, stack);
    while not empty(stack) do
        x :=pop(stack);
        foreach y in Γ_dist(x,k) do
            touched := revise(x,y);
            if touched then
                {at least one value is removed
                from the domain of y};
                if currentDomain_y = ∅ then
                    return false
                push(y, stack);
    return true
end
```

5.1 Résultats préliminaires

Nous avons montré dans la section précédente que Forward-Checking peut être meilleur que MAC sur des problèmes aléatoires. Dans cette section, nous présentons quelques résultats préliminaires sur l'utilité d'appliquer la procédure $MAC(dist_k)$ sur des CSP binaires. Sachant que la distance entre les paires de variables dépend de la densité du graphes des contraintes, nous avons étudié des CSP dont le graphe est peu dense. Sur de tels problèmes, MAC est meilleur que FC. Aussi, on peut tester $MAC(dist_k)$ pour plusieurs valeurs de k (de 1 jusqu'à la distance maximum entre les paires de variables). Notons que le CSP le plus contraint est celui dont le graphe de contraintes est complet. Pour un tel CSP, la distance maximum est égale à 1.

Nous avons effectué des expérimentations sur la même classe présentée dans section 4, c'est-à-dire la classe $< 90, 20, 272, 222 >$. La distance maximale de cette classe est égale à 5.

La table 4 contient les résultats sur la performance de l'algorithme générique $MAC(dist_k)$ et cela pour toutes les distances. Aussi, nous rappelons la performance de FC et MAC sur la même classe. Il est clair que MAC est meilleur que FC sur cette classe. Dans la table 4, nous présentons trois lignes par procédure. Ces lignes montrent les résultats sur les problèmes consistants (sur les 20 CSP testés) et sur ceux inconsistants. Nous montrons également une vue globale sur l'ensemble des problèmes.

Nous pouvons remarquer que pour $distance = 1$, la procédure $MAC(dist_1)$ est moins performant que FC (et MAC). Ceci peut être expliqué par le fait que $MAC(dist_1)$ établit la consistance d'arc sur le voisinage de la variable courante. En d'autres termes, l'algorithme examine toutes les variables connectées à la variables en cours d'affectation. Par exemple, voir la figure 1, la contrainte C_{wy} entre les variables w et y est révisée et la consistance d'arc est établie sur cette contrainte. Cet effort n'est pas réalisé par FC qui effectue une forme restreinte d'arc-consistance directionnelle par rapport à son voisinage. Il semble que l'application de $MAC(dist_1)$ n'est pas profitable.

Concernant les autres distances, nous remarquons que $MAC(dist_k)$ (pour k=2 à 5) améliore l'algorithme MAC (et FC). Les meilleurs résultats sont obtenus pour $k = 2$, $MAC(dist_2)$ est meilleur que FC et MAC par rapport au temps d'exécution et au nombre de tests de consistance.

Nous soulignons que pour les distance supérieures à 5, on retrouve l'algorithme MAC. En effet, pour ces distances toutes les variables connectées à la variable courante (soit x cette variable) appartiennent à $\Gamma_{dist}(x, k)$.

Le comportement de $MAC(dist_k)$ semble robuste sur les CSP consistants et inconsistants. Nous remarquons que l'efficacité de cet algorithme varie selon la distance considérée. Nous signalons que , quand la distance augmente (et quand elle atteint la distance maximum), le comportement de $MAC(dist_k)$ rejoint celui de MAC.

Les résultats dans la table 4, montrent que la meilleure distance à appliquer sur les classes testées est 2. Ceci démontre que sur les CSP peu denses, il existe un niveau intermédiaire de consistance d'arc dont le maintien pendant la recherche améliore l'efficacité des approches prospectives.

6 Discussion et travaux futurs

Les résultats obtenus dans cet article montrent que l'utilité du maintien de la consistance locale pendant la recherche dépend de la structure du problème à résoudre. Ces observations expérimentales ouvrent de nouvelles perspectives sur l'intégration des méthodes de filtrage basées sur la consistance locale dans les algorithmes de résolution. Nous envisageons d'étendre ce travail dans les directions suivantes :

- l'adaptation dynamique du niveau de consistance locale à appliquer pendant la recherche. Ceci pourrait être réalisé en exploitant les caractéristiques des instances traitées.
- l'intégration d'autres formes de consistances locales pendant la recherche. Par exemple, l'arc-consistance directionnelle et la chemin-consistance directionnelle et restreinte.
- l'analyse dynamique (apprentissage) du processus

Procédure		#checks(thousands)	#nodes	CPU time(sec.)
FC	Sat (12/20)	764051	13515952	56.21
	Unsat (8/20)	1393907	24848571	102.92
	All	1015993	18049000	74.89
MAC($dist_1$)	Sat (12/20)	650916	10468120	89.27
	Unsat (8/20)	1204855	19150580	163.67
	All	872492	13941104	119.03
MAC($dist_2$)	Sat (12/20)	144314	315024	8.02
	Unsat (8/20)	284703	600569	15.86
	All	200469	429242	11.16
MAC($dist_3$)	Sat (12/20)	203087	44809	8.44
	Unsat (8/20)	403961	79971	15.93
	All	283437	58874	11.44
MAC($dist_4$)	Sat (12/20)	232389	38879	8.74
	Unsat (8/20)	460891	70606	17.58
	All	323790	51570	12.28
MAC($dist_5$)	Sat (12/20)	232907	39116	8.73
	Unsat (8/20)	461776	70913	17.28
	All	324454	51835	12.15
MAC	Sat (12/20)	232860	41153	9.11
	Unsat (8/20)	459242	75253	17.74
	All	323412	54793	12.56

TAB. 4 – Comparaison entre FC, MAC et MAC($dist_k$) on <90, 20, 272, 222>

de propagation afin de déterminer où et quand il faut appliquer la consistance locale.

7 Conclusion

Dans cet article, nous avons montré que le niveau de filtrage par consistance d'arc à appliquer pendant la recherche dépend des caractéristiques des problèmes testés. Les expérimentations montrent que FC est plus performant que MAC sur des CSP difficiles avec un graphe de contraintes dense et une dureté faible tandis que MAC est meilleur sur les CSP peu dense avec une dureté de contraintes élevée. Par conséquent, aucun des algorithmes se montrent le meilleur sur tout type de CSP. Un autre résultat intéressant dans ce papier est la proposition d'un algorithme générique de résolution de CSP qui adapte le niveau de consistance locale à appliquer selon un paramètre précisant la distance. Les expérimentations montrent que sur certaines classes de CSP, il existe une valeur de la distance pour laquelle $MAC(dist_k)$ est meilleur que FC et MAC.

Références

[1] F. Bacchus. Extending forward checking. In *Proceedings of CP'2000*, pages 35–51, 2000.

[2] Roberto J. Bayardo Jr. and Robert C. Schrag. Using csp look-back techniques to solve real-world sat instances. pages 203–208, Providence (Rhode Island, USA), July 1997.

[3] C. Bessiere. Arc consistency and arc consistency again. *Artificial Intelligence*, 65 :179–190, 1994.

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

[5] C. Bessiere and J. Regin. MAC and combined heuristics : two reasons to forsake FC (and CBJ ?) on hard problems. In *Proceedings of CP'96*, pages 61–75, 1996.

[6] J.R. Binter and E. Reingold. Backtracking programming techniques. *Communications of the ACM*, 18(11) :651–656, 1975.

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

[8] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[9] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. volume 1, pages 248–253, Seattle, Washington (USA), August 4–10 2001.

[10] D. Frost, R. Dechter, C. Bessière, and J.C. Régin. Random uniform CSP generators. http ://www.lirmm.fr/ bessiere/generator.html, 1996.

[11] J. Gaschnig. Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisfying assignments problems. In *Canadian Artificial Intelligence Conference*, pages 268–277, 1978.

[12] M. Ginsberg. Dynamic backtracking. *Artificial Intelligence*, 1 :25–46, 1993.

[13] S.W. Golomb and L.D. Baumert. Backtrack programming. *JACM*, 12 :516–524, 1965.

[14] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. pages 366–371, Nagoya (Japan), August 1997.

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

[16] P. Prosser. Hybrid algorithms for the constraint satisfaction problems. *Computational Intelligence*, 9(3) :268–299, 1993.

[17] D. Sabin and E. freuder. Contradicting conventional wisdom in constraint satisfaction. In *ECAI'94*, 1994.

[18] D. Sabin and E. C. Freuder. Understanding and improving the mac algorithm. In *CP'97*, 1997.

[19] J.P.M. Silva and K.A. Sakallah. Grasp - a new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, 1996.

[20] B.M. Smith and S.A. Grant. Trying harder to fail first. In *Proceedings of ECAI'98*, pages 249–253, Brighton, UK, 1998.

[21] M.R.C. van Dongen. $AC3_d$ an efficient arc consistency algorithm with a low space complexity. In *Proceedings of CP'02*, pages 755–760, 2002.

[22] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedigns of ICCAD'2001*, pages 279–285, San Jose, CA (USA), November 2001.