Amélioration par apprentissage de la recherche à divergences limitées

Wafa Karoui, Marie-José Huguet, Pierre Lopez, Wady Naanaa

Amélioration par apprentissage de la recherche à divergences limitées

Wafa Karoui [1] **Marie-José Huguet** [2] **Pierre Lopez** [2] **Wady Naanaa** [3]

[1] Faculté des Sciences de Tunis, Campus Universitaire, 2092 Manar II, Tunisie
[2] LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse cedex 4
[3] Faculté des Sciences de Monastir, Boulevard de l'environnement, 5019 Monastir, Tunisie
wafa.karoui@laposte.net {huguet,lopez}@laas.fr naanaa.wady@planet.tn

Résumé

Dans le cadre de la résolution des problèmes de satisfaction de contraintes (CSP), nous présentons une nouvelle méthode intitulée *Minimal Discrepancy Search* (MDS), basée sur le principe de la méthode *Limited Discrepancy Search* (LDS) proposée dans [11]. Cette nouvelle méthode s'appuie sur deux extensions de LDS : l'une consistant à tirer profit des échecs rencontrés lors des instanciations ; l'autre permettant de limiter le nombre de divergences à mettre en œuvre et ainsi accélérer la détection de problèmes insolubles. Une amélioration de MDS, *MDS(i)*, permet d'incrémenter le nombre de divergences d'un pas donné i.

Pour attester de l'efficacité de MDS et de MDS(2), des comparaisons avec des méthodes de résolution telles que *Forward Checking* (FC) et *Maintening Arc-Consistency* (MAC) ont été menées. Dans toutes nos expérimentations, MDS s'avère compétitive par rapport à FC et MAC ; par ailleurs, MDS(2) surclasse les autres méthodes.

Abstract

In the context of solving constraint satisfaction problems (CSPs), we present in this paper a new method called Minimal Discrepancy Search (MDS) which is an improvement of Limited Discrepancy Search (LDS) proposed in [11]. This new method is based on two extensions of LDS. The first one take benefit from fails during the search. The second one limits the number of discrepancies needed to solve a problem; it leads to speed up the detection of untractable problems. An improvement of MDS, *MDS(i)*, is also proposed. In MDS(i), the number of discrepancies increases following a given step i. To validate MDS and MDS(2), we compare them with other solving methods such as *Forward Checking* (FC) and *Maintening Arc-Consistency* (MAC). In all experiments the results obtained by MDS are close from those obtained with FC and MAC, whereas MDS(2) outperforms all the other methods.

1 Introduction

Plusieurs méthodes de recherche arborescente comme LDS sont basées sur le concept de *divergence* (ou écart) par rapport aux choix d'une heuristique de parcours [11, 13, 16, 19, 22]. Le point commun implicite entre toutes ces méthodes est leur désir de limiter autant que possible ces divergences en chemin vers une éventuelle solution, afin d'accélérer la recherche.

Le travail présenté dans cet article expose une nouvelle variante de LDS. Celle-ci essaye d'exploiter les échecs rencontrés lors de l'exploration de l'arbre de recherche pour la diriger vers les parties de l'arbre qui contiennent des solutions, de manière à éviter les divergences. Plus précisément, chaque fois que l'algorithme de résolution rencontre un échec en affectant une valeur à une variable, cet échec est utilisé dans la suite pour guider les choix de l'heuristique et privilégier les instanciations qui expliquent l'échec rencontré.

Cet article est organisé de la façon suivante. Dans le paragraphe suivant, nous introduisons les notions de propagation de contraintes et de recherche arborescente tronquée. Puis, nous rappelons le principe de la méthode originelle LDS et l'apport des variantes les plus connues. Nous introduisons ensuite l'amélioration MDS que nous proposons et nous présentons son principe suivi des justifications de validité. Pour finir, nous présentons l'algorithme et les résultats obtenus.

2 Stratégies de résolution existantes

Face au caractère combinatoire des problèmes de satisfaction de contraintes (CSP), il est nécessaire d'introduire des techniques de filtrage de valeurs des variables inconsistantes (propagation de contraintes), ainsi que des méthodes de résolution basées sur des heuristiques de parcours de l'arbre de recherche de valeurs pour les variables.

2.1 Propagation de contraintes

Les algorithmes de filtrage s'appuient sur des techniques de propagation de contraintes, notamment la cohérence de sommets et la cohérence d'arcs, tout CSP admettant un CSP binaire équivalent. Plusieurs algorithmes de cohérence d'arc (AC) ont été proposés. Depuis 1977 où Macworth proposa AC-1 et surtout AC-3 [15], de nombreuses améliorations ont suivi avec comme objectif la réduction des complexités spatiale et/ou temporelle : AC-4, introduisant la notion de support, puis AC-6 et AC-7, ont permis d'atteindre des complexités intéressantes (*e.g.*, pour AC-6, $O(ed^2)$ en temps et $O(ed)$ en espace). En 2001, Bessière et Régin proposent AC-2000 et AC-2001 basés sur AC-3, alliant simplicité et efficacité [4]. Ces deux algorithmes ciblent toutefois des problèmes relativement peu contraints, leur efficacité étant mise en défaut pour des propagations en trop grand nombre. La même année Zhang et Yap proposent AC-3.1, autre amélioration de AC-3, simple, efficace, et d'application plus large [23].

En 2005, Bessière *et al.* proposent AC-2001/3.1 [5] qui a la particularité d'avoir une complexité temporelle dans le pire des cas optimale, tout en restant simple. Les expérimentations montrent que cet algorithme est compétitif avec AC-6.

2.2 Les algorithmes d'instanciation FC et MAC

FC et MAC sont des algorithmes de résolution de CSP utilisant la propagation de contraintes. Généralement, les algorithmes de type AC sont évalués dans MAC (*Maintening Arc-Consistency* [20]), algorithme d'instanciation très efficace. Améliorer MAC a donc longtemps été synonyme d'amélioration de AC. En 2004, Likitvivatanavong *et al.* proposent d'étudier des stratégies plus performantes de MAC, indépendamment de toute amélioration de AC [14]. Dans ce cadre, ils proposent *mac3cache*, amélioration de mac3 (*i.e.*, MAC basé sur AC-3) s'appuyant sur l'idée d'un cache et conservant ainsi toute la simplicité de AC-3. Ils proposent également deux autres versions, *mac3.1residue* et *mac3.1resOpt*, basés sur une notion de "résidu du support". En raison de structures de données supplémentaires de ces deux dernières versions, celles-ci semblent s'effacer au bénéfice de mac3cache.

MAC reste cependant un algorithme trop coûteux pour les problèmes denses et ayant des contraintes de faible dureté, pour lesquels les traitements de propagation nécessaires ne sont pas compensés pas une réduction suffisante de l'espace de recherche. Pour ces problèmes, FC reste plus performant. L'intérêt de MAC est plus flagrant pour des problèmes peu denses et ayant des contraintes dures [6, 10].

Ainsi, dans l'évaluation de la méthode proposée, on se comparera à FC pour les problèmes de la première catégorie et à une variante efficace de MAC (*mac3cache*) pour les problèmes de la seconde catégorie.

3 La méthode proposée : Minimal Discrepancy Search

3.1 Rappel sur le principe de LDS

3.1.1 Principe de LDS

LDS est une méthode de recherche de solution basée sur un parcours particulier d'un arbre de recherche. A partir d'une heuristique d'instanciation donnant un ordre sur les variables du problème et un ordre sur les valeurs des variables, une "solution" initiale est générée. Si cette "solution" s'avère être un fait non admissible (ne respectant pas les contraintes du problème), on autorise la méthode à ne pas suivre les choix de l'heuristique d'instanciation pour certaines variables. Le fait de s'écarter des choix guidés par l'heuristique correspond à la notion de divergence.

3.1.2 Notion de divergence

La notion de divergence peut être définie comme étant la contradiction des choix d'une heuristique d'instanciation. En effet, une heuristique est par définition susceptible de commettre des erreurs et il se révèle donc être parfois avantageux de ne pas la suivre.

Plusieurs algorithmes se sont servis de cette notion afin d'éviter les "early mistakes" et de donner de nouvelles orientations au parcours de l'espace de recherche en autorisant un certain nombre de divergences. Le nombre de divergences permises représente le nombre de fois où l'on s'écarte du premier choix de l'heuristique pour la recherche d'une solution admissible. La notion de divergences a été définie initialement pour les arbres binaires (*i.e.*, à variables binaires) et a ensuite été adaptée pour des arbres non binaires.

Dans le cas d'un arbre binaire, on suppose que, à chaque nœud, le fils gauche représente le premier choix

de l'heuristique (pas de divergence) et le fils droit représente une divergence. Dans le cas d'un arbre n-aire, on suppose qu'à chaque nœud le fils le plus à gauche représente le premier choix de l'heuristique, tous les autres fils correspondant à des divergences. Dans le reste de cet article, le deuxième fils gauche correspond à une divergence, le troisième fils gauche correspond à deux divergences, etc.

Suivant ce principe, si on n'obtient pas de solutions par les premiers choix de l'heuristique, on se permet une divergence, puis deux si une solution admissible n'est toujours pas obtenue, et ainsi de suite jusqu'à ce qu'on trouve une solution ou qu'on atteint le maximum de divergences possibles (tout l'arbre dans le pire des cas) (algorithme 1).

Algorithm 1 LDS(X, D, C, k-max, Sol)

```
1: k ← 0
2: Sol ← NIL
3: tant que (Sol = NIL) et (k ≤ k-max) faire
4:    Sol ← LDS-itération(X, D, C, k, Sol)
5:    k ← k+1
6: fin tant que
7: retourne Sol
```

Algorithm 2 LDS-itération(X, D, C, k, Sol)

```
1: si X = ∅ alors
2:    retourne Sol
3: sinon
4:    x_i ← First_VariableOrdering(X)
5:    v_i ← First_ValueOrdering(D_i, k)
6:    si v_i ≠ premier fils gauche alors
7:       k ← k−1    % on fait une divergence
8:    fin si
9:    si Sol ⋃ {(x_i, v_i)} satisfait C alors
10:      D' ← Update(X\{x_i}, D, C, (x_i, v_i))
11:      si aucun domaine de D' n'est vide alors
12:         Sol ← LDS-itération(X\{x_i}, D, C, k, Sol
            ⋃ {(x_i, v_i)})
13:      fin si
14:   sinon
15:      retourne NIL
16:   fin si
17: fin si
```

VariableOrdering() correspond à l'heuristique de choix des variables (*min-domain* dans notre cas ; voir plus loin) ; *ValueOrdering()* correspond à l'heuristique de choix des valeurs (*min-conflict*) ; *Update()* est la fonction d'actualisation de FC (filtrage des valeurs des domaines des variables voisines de la dernière variable instanciée).

3.1.3 Limites de LDS

Le premier inconvénient de LDS est qu'elle présente trop de redondances dans le parcours de l'arbre de recherche. En effet, pour un nombre donné de divergences, on énumère de 0 jusqu'à ce nombre quand on applique son itération. Pour éviter ces redondances, la recherche à divergences limitées améliorée (ILDS) s'avère efficace pour les arbres binaires [13]. Sa généralisation à des arbres n-aires ne l'est toutefois pas autant.

Son second inconvénient se rencontre lorsque le problème est insoluble où l'on est obligé de parcourir tout l'espace de recherche, et donc de toujours autoriser le maximum de divergences à la dernière itération, avant de déduire l'infaisabilité du problème.

Une autre amélioration de LDS est la recherche à divergences limitées en profondeur (DDS) [22] qui favorise les divergences dans la partie haute de l'arbre en premier.

3.1.4 LDS avec propagation du type Forward Checking

La méthode LDS proposée initialement ne fait pas intervenir de techniques de propagation. Pour la résolution de CSP, on peut envisager d'intégrer des propagations du type de celles présentes dans Forward-Checking. Dans *LDS-itération*, à chaque fois qu'une variable est instanciée, on pourrait ainsi considérer l'impact de l'instanciation d'une variable sur les variables non encore instanciées et reliées par une contrainte à cette variable instanciée, donc son voisinage. Dans le reste de cet article, on appellera LDS la version de LDS qui intègre cette propagation au voisinage d'une variable instanciée. De plus, l'heuristique sur l'ordre des variables que l'on a choisie d'utiliser est *min-domain* et l'heuristique sur l'ordre des valeurs est *min-conflict*, ceci pour tous les algorithmes évoqués dans la suite (MDS, FC, MAC).

3.2 Permuted Limited Discrepancy Search (PLDS)

Cette amélioration de LDS a pour objectif de remédier à son principal inconvénient qui est de faire trop de redondances dans le parcours. Dans LDS, l'unique responsable du sens de parcours de l'espace de recherche est l'heuristique sur l'ordre des variables. Ceci implique que lorsqu'on réitère LDS en incrémentant le nombre de divergences autorisées, on se retrouve fréquemment avec la même variable à instancier. Supposons que cette variable soit justement celle qui élimine des valeurs nécessaires à la résolution, le développement de sa branche est alors inutile.

Pour remédier à ce genre de situations, on associe une priorité, initialement nulle, à chaque variable. Cette priorité est incrémentée chaque fois que cette variable essuie un échec, c'est-à-dire lorsqu'elle est la dernière variable à devoir être instanciée dans une branche et que son domaine est vide. Dans les parcours qui vont suivre, cette variable sera prioritaire et sera placée en tête de branche. On évitera ainsi de se retrouver dans une situation d'incohérence pour cette variable. Ainsi, le choix des variables à instancier sera basé en premier lieu sur l'heuristique *min-domain*, en second lieu sur un ordre de priorité et en troisième lieu sur l'ordre des indices.

En fait, en introduisant la notion de priorité, nous visons une correction de l'heuristique *min-domain* en la guidant vers les variables concrètement contraintes ; ces variables influencent grandement la recherche de solutions. On corrige donc les erreurs de l'heuristique en ajoutant cette notion de priorité qui n'est autre qu'une forme d'apprentissage tout au long du déroulement de l'algorithme de résolution. Cela permet d'exploiter les échecs passés et d'en tirer des informations exploitables dans la suite du déroulement. Pour accélérer le parcours, il s'agit ainsi de faire remonter à la surface de l'espace de recherche, et donc parmi les premières variables à instancier quand on réitère LDS, les sous-problèmes difficiles et les sous-problèmes surcontraints.

La version de LDS qui tire profit de cette notion de réordonnancement des variables est appelée dans la suite *Permuted LDS* (PLDS). PLDS ne peut pas regénérer la même branche avant d'incrémenter la priorité de toutes les variables présentes sur cette branche pour aboutir au même ordre. Ceci implique que la redondance est diminuée d'au moins un facteur égal au nombre de variables.

Prenons l'exemple d'un CSP comportant quatre variables. Supposons que le premier ordre proposé par l'heuristique a mené vers un échec sur une variable. Ce même ordre ne pourra être retrouvé que lorsqu'un échec aura été détecté sur les trois autres variables. Ainsi, dans un CSP de quatre variables, deux branches identiques se trouvent séparées d'au moins trois branches.

Si entre ces deux branches représentant le même ordre, une variable incrémente deux fois sa priorité, alors toutes les autres variables devraient faire de même pour aboutir à une égalité de priorité et on devra compter deux fois le nombre des variables avant de retrouver la branche initiale. Ainsi, le facteur de redondance est dans le pire des cas divisé par le nombre de variables ou en général d'un multiple de ce nombre.

3.3 Restricted Discrepancy Search (RDS)

Cette amélioration de LDS vient remédier à son inconvénient de parcourir tout l'espace de recherche dans le cas de problèmes insolubles.

En effet, lorsqu'on fixe un nombre de divergences autorisées pour la méthode LDS, on peut consommer totalement ou non ces divergences. Si l'ensemble des divergences permises est consommé et qu'aucune solution n'est trouvée, une nouvelle itération de LDS se déroulera en incrémentant le nombre de divergences autorisées. En revanche, si l'ensemble des divergences permises n'est pas consommé, il n'est pas nécessaire de continuer à relancer LDS avec un nombre supérieur de divergences, même si aucune solution n'a été trouvée car il ne sera pas possible d'aller plus loin dans le parcours de l'arborescence (le problème est alors insoluble). Relancer LDS reviendrait ici à développer le même arbre de recherche qu'à une itération précédente. Ainsi, on peut arrêter l'incrémentation du nombre de divergences autorisées dès qu'on constate que ce nombre n'a pas été totalement consommé.

La version de LDS qui tire profit de cette notion de restriction du parcours au seul nombre de divergences nécessaires est appelée dans la suite *Restricted DS* (RDS).

3.4 Minimal Discrepancy Search

La méthode MDS est la superposition des deux extensions PLDS et RDS. Sa validité provient de celle des deux techniques composantes. PLDS se distingue de LDS par un nouveau sens de parcours de l'espace de recherche. Cette technique intervient dans l'accélération de la résolution, mais reste toujours une méthode de résolution complète, qui trouve une solution si elle existe. RDS est une technique qui nous épargne d'incrémenter le nombre de divergences plus que nécessaire, sans pour autant passer à côté de solutions.

Le fonctionnement de la méthode MDS est donné par l'algorithme 3.

Algorithm 3 MDS(X, D, C, k-max, Sol)

```
1:  k ← 0
2:  Sol ← NIL
3:  tant que (Sol = NIL) et (k ≤ k-max) faire
4:    si TestArrêt() alors
5:      exit
6:    sinon
7:      Sol ← MDS-itération(X, D, C, k, Sol)
8:      k ← k+1
9:    fin si
10: fin tant que
11: retourne Sol
```

Algorithm 4 MDS-itération(X, D, C, k, Sol)

```
1: si X = ∅ alors
2:    retourne Sol
3: sinon
4:    x_i ← First_VariableOrdering(X)
5:    v_i ← First_ValueOrdering(D_i, k)
6:    si v_i ≠ premier fils gauche alors
7:       si v_i ≠ NIL alors
8:          k ← k-1  % on fait une divergence
9:       sinon
10:         Priorité(x_i) ← Priorité(x_i)+1
11:      fin si
12:   fin si
13:   si v_i ≠ NIL alors
14:      si Sol ⋃ {(x_i, v_i)} satisfait C alors
15:         D' ← Update(X\{x_i}, D, C, (x_i, v_i))
16:         si aucun domaine de D' n'est vide alors
17:            Sol ← MDS-itération(X\{x_i}, D, C, k,
               Sol ⋃ {(x_i, v_i)})
18:         fin si
19:      sinon
20:         retourne NIL
21:      fin si
22:   fin si
23: fin si
```

TestArrêt() est le test qui retourne vrai si la dernière itération n'a pas été interrompue par dépassement de la borne autorisée sur le nombre de divergences ; le problème est ainsi insoluble. Si le test retourne faux, on incrémente le nombre de divergences autorisées pour continuer la recherche.

Ainsi, le principe de la nouvelle méthode MDS est exactement celui de LDS, c'est-à-dire qu'elle examine en premier les branches de l'arbre qui accumulent le plus petit nombre de divergences. La première différence est que chaque variable est dotée d'une priorité qui est initialement la même pour toutes les variables et qu'à chaque fois que l'une d'entre elles échoue, sa priorité est incrémentée pour guider les prochains choix de l'heuristique. La seconde différence est que le nombre de divergences n'est plus incrémenté aveuglément jusqu'au maximum du nombre de divergences permises par l'arbre de recherche, mais tient compte du problème à résoudre. Ainsi, la nouvelle méthode consomme moins de divergences que LDS et ILDS, d'où son nom de *Minimal* Discrepancy Search.

3.5 Application à un exemple

Soit le CSP P(X, D, C) défini par :
X={x0, x1, x2} ;
D={D0, D1, D2} avec D0=D1=D2={0, 1, 2, 3, 4} ;
C={(x0, 0), (x1, 4)}∪{(x0, 0), (x2, 4)}∪{(x0, 1), (x1, 4)}∪{(x0, 1), (x2, 4)}∪{(x0, 2), (x1, 4)}∪{(x0, 2), (x2, 4)}∪{(x0, 3), (x1, 4)}∪{(x0, 3), (x2, 4)}∪{(x0, 4), (x2, 2)}∪{(x0, 4), (x2, 3)}∪{(x1, 0), (x2, 0)}∪{(x1, 0), (x2, 1)}∪{(x1, 0), (x2, 2)}∪{(x1, 0), (x2, 3)}∪{(x1, 1), (x2, 0)}∪{(x1, 1), (x2, 1)}∪{(x1, 1), (x2, 2)}∪{(x1, 1), (x2, 3)}∪{(x1, 2), (x2, 0)}∪{(x1, 2), (x2, 1)}∪{(x1, 2), (x2, 2)}∪{(x1, 2), (x2, 3)}∪{(x1, 3), (x2, 0)}∪{(x1, 3), (x2, 1)}∪{(x1, 3), (x2, 2)}∪{(x1, 3), (x2, 3)}.

On se propose d'appliquer l'algorithme FC pour résoudre ce CSP. L'arbre de recherche obtenu compte 44 nœuds développés (NND) comme cela est représenté sur la figure 1.

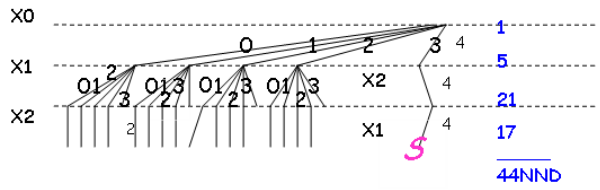

FIG. 1 – *Arbre développé par Forward-Checking*

En appliquant l'algorithme LDS, on obtient un arbre de recherche encore plus développé contenant 95 nœuds.

En revanche, en appliquant MDS, on obtient un arbre de recherche de 9 nœuds puisque l'incrémentation de la priorité de la variable x2 est très favorable (figure 2).

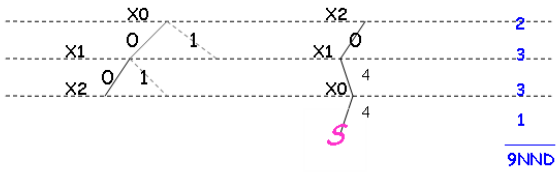

FIG. 2 – *Arbre développé par Minimal Discrepancy Search*

3.6 Amélioration de Minimal Discrepancy Search

Une dernière amélioration de MDS que nous proposons, appelée *MDS(i)*, consiste à incrémenter le nombre de divergences autorisées d'un certain pas i supérieur ou égal à 1. De ce fait, MDS peut être considéré comme un cas particulier de MDS(i) avec un pas régulier toujours égal à un.

4 Résultats

4.1 Cadre de l'expérimentation

L'expérimentation a porté sur des CSP aléatoires de diverse taille. Un CSP aléatoire est caractérisé par quatre paramètres n, d, p1 et p2, où n est le nombre de variables du problème, d est le nombre de valeurs par domaine, p1 est la probabilité d'existence d'une contrainte entre deux variables (densité du graphe des contraintes) et p2 donne la proportion approximative de couples de valeurs incompatibles dans la définition des contraintes (dûreté des contraintes). Pour des valeurs fixées de n, d et p1, il existe une zone de variation de p2 où les CSP sont particulièrement difficiles à résoudre (pic de complexité). Les instances de problèmes considérées dans nos expérimentations ont été choisies dans cette zone critique. Pour chaque instance de CSP aléatoire, *i.e.,* pour chaque quadruplet (n, d, p1, p2), on génère 500 problèmes, considérant que 500 est un nombre suffisant pour atténuer l'effet des perturbations dues à des cas particuliers. Le générateur utilisé est du type désigné par modèle B dans la littérature [7, 8, 9]. Les performances sont évaluées en termes de temps et de nombre de nœuds développés dans l'arbre de recherche (NND).

4.2 Résultats expérimentaux de RDS, PLDS, MDS et MDS(i)

L'algorithme de recherche à divergences minimales (MDS) se distingue par le nombre minimal de divergences nécessaires avant d'obtenir une solution (*nbDiv*) que ce soit sur les problèmes solubles ou surcontraints. La comparaison avec RDS et PLDS sur les CSP <500-20-20-0.5>, <500-40-10-0.2> et <500-60-05-0.2> atteste ce résultat.

Etant la composition de deux améliorations de LDS, MDS est globalement plus performante que LDS (avec propagation), que ce soit en nombre de nœuds développés ou en temps d'exécution, sur des problèmes solubles ou insolubles.

Les courbes de la figure 3 illustrent une comparaison de MDS avec PLDS et RDS sur les instances citées ci-dessus, en termes de nombre de divergences consommées pour la résolution de problèmes solubles et insolubles. Les courbes de la figure 4 permettent, elles, de comparer MDS et LDS avec propagation, en termes de temps d'exécution et de nombre de nœuds développés.

Les courbes de la figure 5 illustrent le comportement de MDS par rapport à FC sur des instances <20-20-0.5>, <60-05-0.5> et <60-5-0.8> (NND et temps pour chaque groupe d'instances). Enfin, les courbes de la figure 6 présentent l'apport de MDS par rapport à MAC sur des instances <40-10-0.2> et <60-5-0.2> (NND et temps pour chaque groupe d'instances).

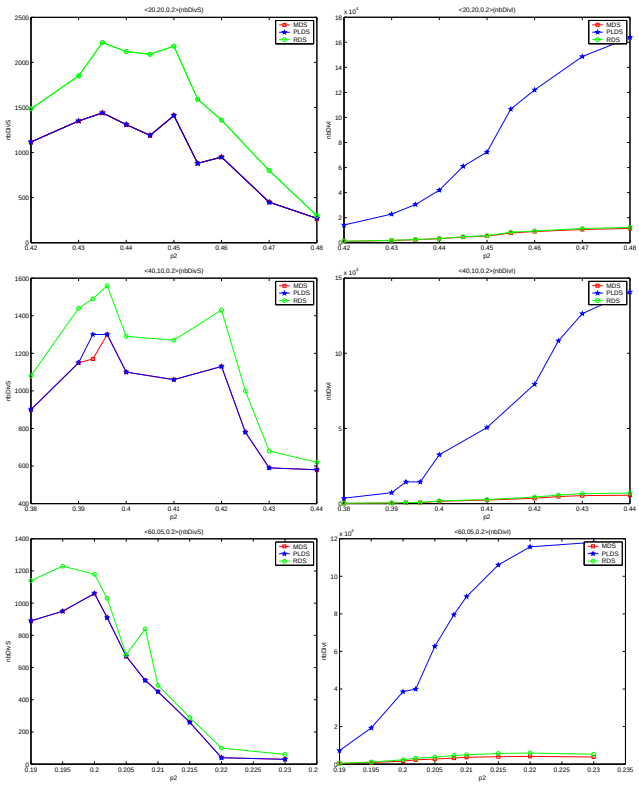

FIG. 3 – *Comparaison de MDS et ses composantes (PLDS et RDS)en terme de nombre de divergences*

Que ce soit selon les comparaisons avec FC sur des problèmes solubles où FC est le plus adéquat pour la résolution ou sur des comparaisons avec MAC sur des problèmes solubles où MAC est le plus performant, MDS(2) présente des résultats très satisfaisants en termes de temps et de nombre de nœuds développés. D'où l'intérêt de cette nouvelle méthode.

5 Conclusion et perspectives

MDS est une méthode complète de résolution de problèmes de satisfaction de contraintes. Elle s'inscrit dans la famille des recherches à divergences limitées (LDS). Les améliorations apportées permettent de pallier les limites d'une méthode LDS de base, à savoir : redondance dans le parcours de l'arbre de recherche et développement complet de l'arbre pour des problèmes insolubles.

La première amélioration, PLDS, est basée sur une exploitation des échecs rencontrés pendant la résolution afin de limiter les redondances dans le parcours. La seconde, RDS, permet de restreindre le nombre de divergences totales permises lorsque les divergences ne sont pas toutes exploitées lors du développement de

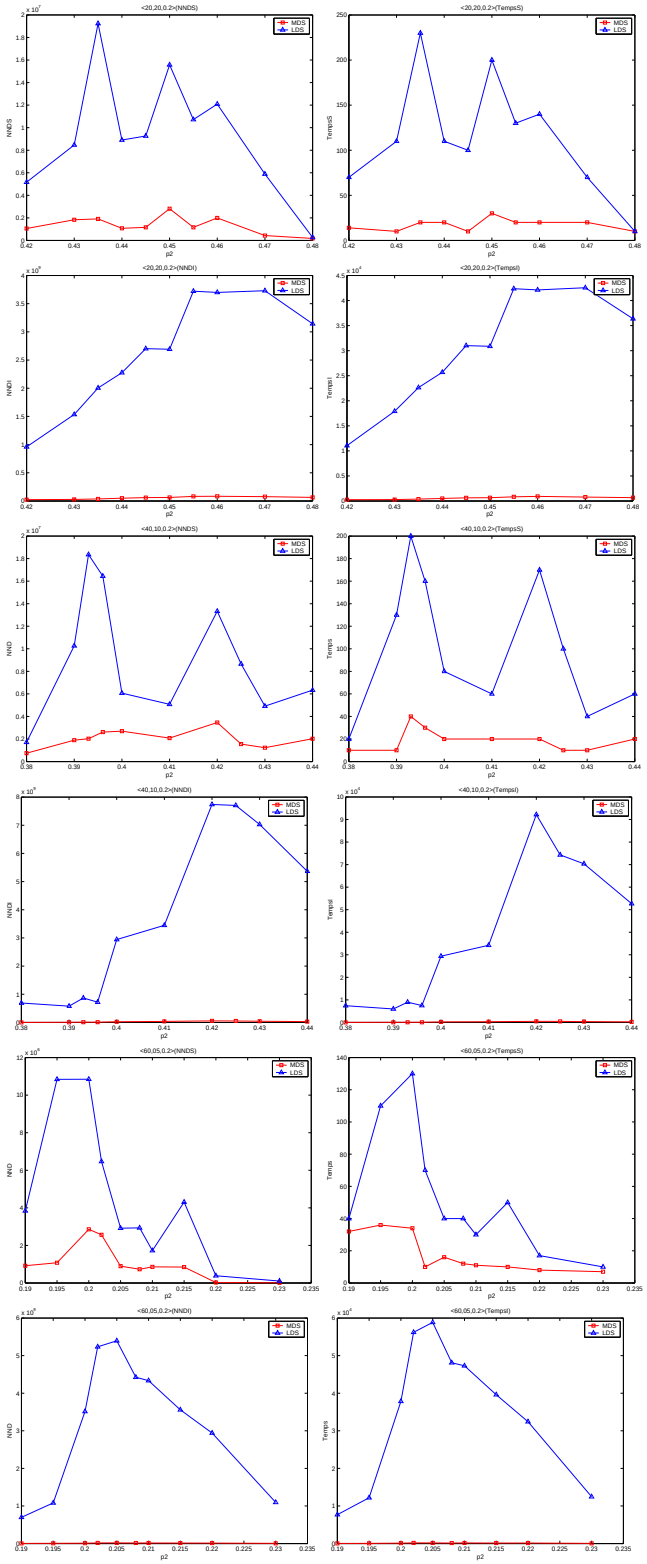

FIG. 4 – *Comparaison de MDS et LDS*

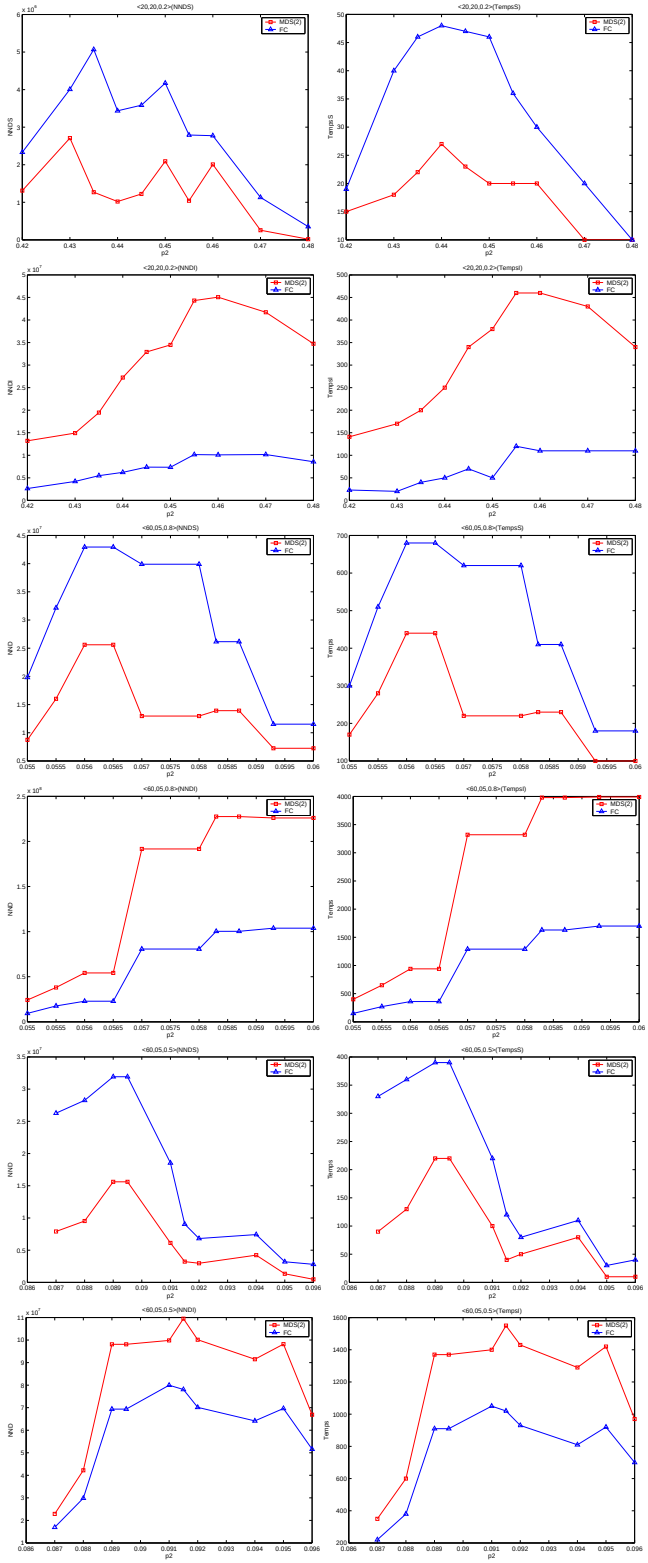

FIG. 5 – *Comparaison de MDS(2) et FC*

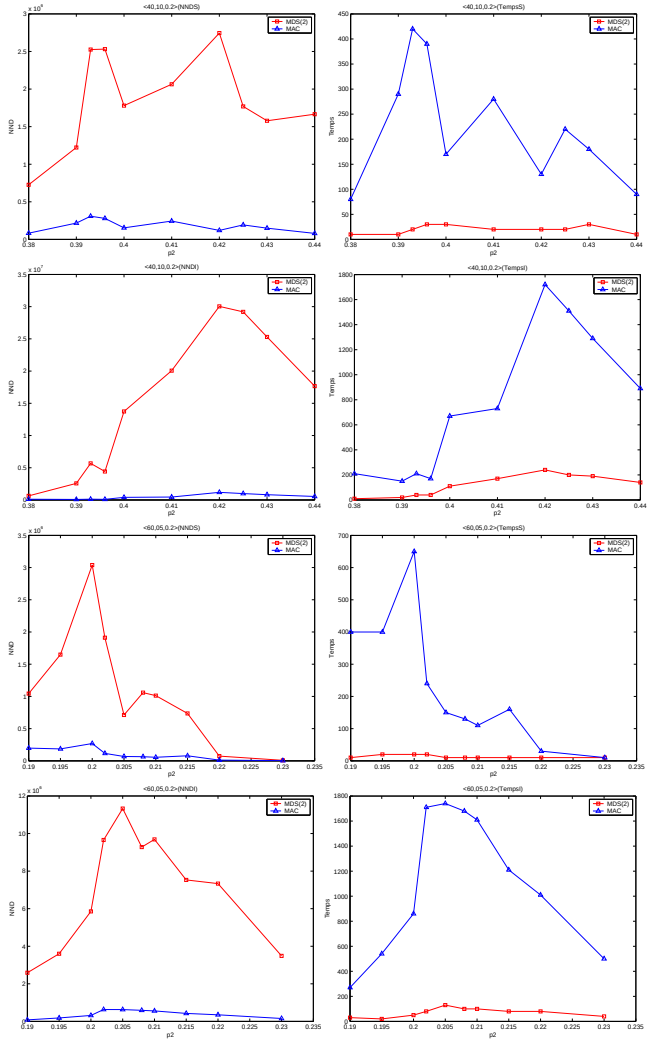

FIG. 6 – *Comparaison de MDS(2) et MAC*

l'arborescence. Ces améliorations procèdent ainsi suivant des techniques d'apprentissage.

Les expérimentations ont montré que PLDS et RDS amélioraient LDS et que la combinaison des deux (méthode MDS) était plus performante que chacune d'elles considérée séparément. De plus, les résultats expérimentaux ont montré que MDS était compétitive avec FC et MAC, en ce qui concerne la résolution de CSP aléatoires, que ce soit en nombre de nœuds développés ou en temps d'exécution.

Enfin, une dernière amélioration, MDS(i), permet d'incrémenter le nombre de divergences autorisé d'un pas donné supérieur ou égal à 1. Les résultats expérimentaux attestent que cette dernière amélioration avec un pas de 2 s'avère nettement plus efficace que MDS, FC et MAC.

Dans l'immédiat, les perspectives de ce travail sont de se comparer à d'autres variantes de MAC (*e.g.*, *mac3residue*) pour vérifier si nos résultats se maintiennent. On pourra aussi envisager de faire varier le pas de MDS(i) et de mener des études sur ce paramètre.

On pourra ensuite envisager une étude expérimentale de MDS sur des problèmes de taille et de complexité beaucoup plus importantes. L'objectif est de montrer l'intérêt opérationnel de MDS dans des contextes réels.

Les méthodes de la famille LDS ont déjà été adaptées à des problèmes d'ordonnancement [21]. En perspective à plus long terme, on pourrait ainsi envisager d'étudier l'impact de notre méthode MDS pour la résolution de ce genre de problèmes, notamment des problèmes d'ordonnancement de projet à moyens limités (RCPSP). Il s'agira alors d'étudier les performances de MDS en regard des méthodes existantes dans ce contexte.

Références

[1] C. Bessière. Arc-consistency and arc-consistency again. *Artificial Intelligence*, 65:179–190, 1994

[2] C. Bessière, A. Chmeiss et L. Saïs. Heuristiques multi-niveaux pour ordonner les variables dans les CSP. In *Proceedings JNPC'01*, pages 49–60, Toulouse, 2001

[3] C. Bessière, E.C. Freuder, and J.-C. Régin. Using constraint metaknowledge to reduce arc-consistency computation. *Artificial Intelligence*, 107:125–148, 1999

[4] C. Bessière and J.-C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings IJCAI-01*, pages 309–315, Seattle, USA, 2001

[5] C. Bessière, J.C. Régin, R. H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consis-

tency algorithm. *Artificial Intelligence*, 2005 (à paraître)

[6] A. Chmeiss and L. Sais. Constraint satisfaction problems: backtrack search revisited. In *Proceedings ICTAI 2004*, pages 252–257, Boca Raton, USA, 2004

[7] I.P. Gent, E. MacIntyre, P. Prosser, B.M. Smith, and T. Walsh. Random constraint satisfaction: flaws ans structures. *Constraints*, 6:345–372, 2001

[8] I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The Constrainedness of Search. In *Proceedings AAAI-96*, Portland, USA, 1996

[9] I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The Scaling of Search Cost. In *Proceedings AAAI-97*,

[10] I.P. Gent and P. Prosser. Inside MAC and FC. APES Research Group Report APES-20-2000, 2000

[11] W.D. Harvey and M.L. Ginsberg. Limited discrepancy search. In *Proceedings IJCAI-95*, pages 607–613, Montréal, Canada, 1995

[12] P. Van Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992

[13] R.E. Korf. Improved limited discrepancy search. In *Proceedings AAAI/IAAI, Vol. 1*, pages 286–291, 1996

[14] C. Likitvivatanavong, Y. Zhang, J. Bowen, and E.C. Freuder. Arc-consistency in MAC: A new perspective. In *Proceedings First International Workshop on Constraint Propagation and Implementation*, Toronto, 2004

[15] A.K. Mackworth and E.C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985

[16] P. Meseguer and T. Walsh. Interleaved and discrepancy based search. In *Proceedings 13^{t}h ECAI*, pages 239–243, Brighton, UK, Wiley, 1998

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

[18] M. Perlin. Arc consistency for factorable relations. *Artificial Intelligence*, 53:329–342, 1992

[19] N. Prcovic. Quelques variantes de LDS. In *Proceedings JNPC'02*, pages 195–208, Nice, 2002

[20] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings PPCP-94*, Seattle, USA, 1994

[21] F. Tercinet. Méthodes arborescentes pour la résolution des problèmes d'ordonnancement, conception d'un outil d'aide au développement. Thèse de Doctorat de l'Université de Tours, 2004

[22] T. Walsh. Depth-bounded discrepancy search. In *Proceedings IJCAI*, pages 1388–1395, 1997

[23] Y. Zhang and R. H.C. Yap. Making AC-3 an optimal algorithm. In *Proceedings IJCAI-01*, pages 316–321, Seattle, USA, 2001