# Intégration partielle de la consistance de chemin dans la résolution des CSP

Assef Chmeiss, Vincent Krawczyk, Lakhdar Saïs

*Actes JFPC 2008*

# Intégration partielle de la consistance de chemin dans la résolution des CSP

**Assef Chmeiss** **Vincent Krawczyk** **Lakhdar Saïs**

Université Lille-Nord de France, Artois, F-62307 Lens
CRIL, F-62307 Lens
CNRS UMR 8188, F-62307 Lens
{chmeiss, krawczyk, sais}@cril.univ-artois.fr

**Résumé**

De nombreuses propriétés de consistance locale ont été exploitées dans la résolution des problèmes de satisfaction de contraintes (CSP). L'objectif est de réduire l'espace de recherche et par conséquent améliorer les méthodes de résolution. La consistance d'arc a été la forme de consistance locale la plus étudiée. Il a été montré que le maintien de la consistance d'arc durant la recherche est très utile pour la résolution des CSP. L'utilisation de formes plus fortes de consistance locale (comme la consistance de chemin) est encore limitée car elles nécessitent la gestion de structures de données complexes.

Dans ce papier, nous proposons une technique permettant l'exploitation, dans la phase de prétraitement, d'une forme partielle de consistance d'arc et de chemin. Cette consistance est basée sur la notion d'intervalles de supports. Nous montrons aussi que, grâce à cette forme de consistance, il est possible de réduire l'espace de recherche pendant la résolution.

## 1 Introduction

Les problèmes de satisfaction de contraintes impliquent l'affectation de valeurs à des variables qui sont soumises à un ensemble de contraintes. Beaucoup d'applications réelles peuvent être formalisées et traitées à l'aide réseaux de contraintes comme le coloriage de graphes, la reconnaissance de formes,... Un CSP est un problème connu pour être NP-complet. La complexité théorique en temps pour résoudre un CSP est exponentielle en la taille du problème. Si $d$ est la taille des domaines (le nombre maximum de valeurs dans un domaine $d_i$) et $n$ est le nombre de variables, la complexité théorique en temps est bornée par la taille de l'espace de recherche, c'est à dire $d^n$. De nombreuses techniques ont été proposées pour améliorer la performance de la recherche classique par retour arrière, telle que la propagation de contraintes, le retour arrière intelligent ou encore la décomposition de réseaux. Les techniques de filtrage se montrent sont très efficaces puisqu'elles réduisent la taille de l'espace de recherche. Pour la résolution des CSP, MAC (Maintaining Arc Consistency) semble être l'algorithme général le plus efficace, il a été proposé par Sabin et Freuder [12]. D'autres approches utilisent les techniques de décomposition basées sur des propriétés structurelles [6][8],ou sur la micro-structure du graphe de contraintes associée au CSP [9].

D'autres travaux concernant la caractérisation de classes traitables ont été proposés, ils exploitent la sémantique des contraintes (contraintes fonctionnelles, ...) ou des propriétés structurelles du graphe (arbres). Nous mentionnons que certaines de ces techniques de filtrages suffisent à résoudre une partie de ces classes. Par exemple, la consistance d'arc est suffisante pour résoudre les CSP dont le graphe de contraintes est un arbre, les contraintes 0/1/All[4] peuvent être résolues en temps polynomial en utilisant la consistance de chemin. Cependant, les filtrages basés sur des propriétés de consistance locale peuvent avoir un effet négatif suivant le type (niveau) de consistance utilisée. Par exemple, le maintien de la consistance de chemin complète pendant la recherche est trop coûteux car nous devons gérer des structures de données complexes. Ce dernier point explique le fait que les travaux sur la consistance de chemin se sont orientés principalement vers des formes restreintes de cette consistance.

Dans ce papier, nous proposons un prétraitement exploitant la notion d'intervalle de supports. Cette notion intégré à des consistances locales comme la consistance d'arc ou de chemin nous permet de réduire le nombre de tests de consistances (voir la section 3). Nous proposons aussi, dans la section 4, une manière (basée sur les intervalles)

d'exploiter une forme partielle de consistance de chemin pour réduire la taille de l'espace de recherche. Le reste du papier est organisé comme suit : la section 2 rappelle des définitions et notations nécessaires pour la suite. Dans la section 3, nous décrivons le prétraitement basé sur les intervalles. La section 4 présente une façon que nous utilisons pour exploiter une forme partielle de consistance de chemin pour améliorer le processus de recherche. Dans la section 5, nous présentons des résultats préliminaires qui montrent que notre approche améliore l'algorithme MAC. Finalement, nous donnons une conclusion sur ce travail dans la section 6.

## 2 Définitions

Un CSP fini (problème de satisfaction de contraintes)[11] est défini comme un triplet $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$. $\mathcal{X}$ est un ensemble fini de $n$ variables $x_1, x_2, \ldots x_n$. $\mathcal{D}$ est un ensemble fini de domaines $d_1, d_2, \ldots d_n$, et $\mathcal{C}$ est un ensemble fini de $m$ contraintes. Pour chaque contrainte $c$ d'arité $k$ avec $\{x_1, \ldots, x_k\} = scope(c)$, nous associons un sous-ensemble du produit Cartésien $(D_{x_1} \times \ldots \times D_{x_k})$ qui est noté $R_c$ et spécifie les valeurs des variables qui sont compatibles entre elles. Nous notons $(v_1, \ldots, v_k) \in R_c$ si et seulement si le tuple satisfait la contrainte $c$.

Dans ce papier nous considérons seulement les CSPs binaires, c'est à dire, les contraintes sont définies sur des paires de variables $\{x, y\}$, cette contrainte sera notée $c_{xy}$. Le voisinage d'une variable $x$, noté $\Gamma(x)$, est l'ensemble des variables qui sont connectés à $x$. Une solution est une affectation de valeurs aux variables qui satisfait toutes les contraintes. Pour un CSP binaire $\mathcal{P}$, le couple $(X, C)$ désigne le graphe appelé le *graphe de contraintes*. Le degré de $x$ dans le graphe de contraintes sera noté $|\Gamma(x)|$. Étant donné un CSP, le problème est de décider si ce CSP possède une solution, de trouver une solution ou de trouver toutes les solutions.

Nous rappelons ci-dessous quelques définitions qui seront utiles dans le reste du papier.

**Définition 1** (consistance d'arc (AC)). *Un domaine $d_i \in \mathcal{D}$ est arc consistant ssi, $\forall a \in d_i, \forall x_j \in \mathcal{X}$ tel que $c_{ij} \in \mathcal{C}$, il existe $b \in d_j$ tel que $(a, b) \in R_{c_{ij}}$. Un CSP est arc consistant ssi $\forall d_i \in \mathcal{D}, d_i \neq \emptyset$ and $d_i$ est arc consistant.*

**Définition 2** (consistance de chemin (PC)). *Une paire de variables $\{x_i, x_j\}$ est chemin consistante ssi $\forall (a, b) \in R_{c_{ij}}, \forall x_k \in \mathcal{X}$, il existe $c \in d_k$ tel que $(a, c) \in R_{c_{ik}}$ et $(b, c) \in R_{c_{jk}}$. Un CSP est chemin consistant ssi $\forall x_i, x_j \in \mathcal{X}$ la paire $\{x_i, x_j\}$ est chemin consistante.*

La consistance de chemin directionnelle (DPC) est une forme affaiblie de la consistance de chemin. Elle est définie par rapport à un ordre sur les variables.

**Définition 3** (consistance de chemin directionnelle[7]). *Un graphe de contraintes est chemin consistant directionnel par rapport à un ordre sur les variables $(x_1, x_2, \ldots, x_n)$, si pour chaque paire de variables $\{x_i, x_j\}$ et chaque couple de valeurs $(a, b) \in R_{c_{ij}}, \forall k > i, j$, il existe $c \in d_k$ tel que $(a, c) \in R_{c_{ik}}$ et $(b, c) \in R_{c_{jk}}$.*

Tandis que le filtrage par AC supprime des valeurs dans les domaines si elles ne sont pas arc consistantes, le filtrage par PC supprime des paires de valeurs si elles ne sont pas chemin consistantes. Pour établir la consistance de chemin complète dans un CSP, le graphe de contraintes doit être complet. Par conséquent, on doit considère la relation universelle entre les couples de variables s'il n'y a pas de contraintes entre elles. Une forme restreinte de PC a été proposée par Debruyne dans [5], appelée consistance de chemin conservative (CPC), qui ne modifie pas la structure du graphe.

## 3 Filtrage basé sur les intervalles

Dans cette section, nous proposons un filtrage basé sur la notion d'intervalles. Ce filtrage sera exploité dans la phase de prétraitement. Notre approche utilise de manière originale le concept d'intervalle de support qui généralise la notion bien connue de premier support dans AC6 [1]. Plus précisément, étant donné une contrainte $c_{xy}$, nous associons à chaque valeur $v$ dans le domaine de $x$ un intervalle constitué du premier et dernier support de $v$ dans le domaine de $y$. Ces intervalles de supports sont calculés lors de la phase de prétraitement en utilisant une adaptation de l'algorithme d'AC appelé consistance d'arc par intervalle (IAC pour Interval Arc Consistency). Notons que IAC ne doit pas être confondue avec la notion de consistance aux bornes (BC) pour les CSPs discrets [10]. En effet, dans BC, les intervalles sont associés aux variables, alors que IAC associe un intervalle à chaque valeur et les tests de consistance sont effectués sur chaque valeur dans les domaines.

Plusieurs objectifs résident dans l'utilisation des intervalles de support. Premièrement, les intervalles de supports calculés lors du premier appel à la consistance d'arc (avant la recherche) sont utilisés pour réduire le nombre de tests de consistance pendant la recherche. Deuxièmement, une forme partielle de consistance de chemin, appelée consistance de chemin par intervalle (Interval Path Consistency), est définie en utilisant des opérations sur les intervalles pour réduire de manière significative la complexité en temps de ce filtrage. Finalement les intervalles de supports rendent possible une intégration efficace d'une forme partielle de consistance de chemin pendant la recherche (voir la section suivante). Avant de présenter notre prétraitement basé sur les intervalles, nous introduisons quelques définitions et propriétés nécessaires.

**Définition 4** (Intervalle de supports). *Soit $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$*

*un CSP binaire.* $\forall x \in \mathcal{X}, \forall y \in \mathcal{X} | c_{xy} \in \mathcal{C}, \forall v \in d_x$, *nous définissons l'intervalle de support associé au couple* $(x, v)$ *par rapport à* $c_{xy}$, *noté* $I_{(x,v),y}$ *comme un intervalle* $[s_{min}, s_{max}]$ *où* $s_{min}$ *(resp.* $s_{max}$*) est le premier (resp. dernier) support de* $v$ *in* $d_y$ *noté* $first(I_{(x,v),y})$ *et* $last(I_{(x,v),y})$.

Nous allons montrons maintenant comment l'intervalle de supports peut être simplement greffé à n'importe quel algorithme de consistance d'arc. En effet, pour une contrainte donnée $c_{xy}$, chaque valeur $v$ de $d_x$ est tout d'abord associée à un intervalle de supports incluant toutes les valeurs de $y$ c.a.d. $I_{(x,v),y} = [v_1, v_{|D_y|}]$ (Algorithm 1, line 4). Deuxièmement, à chaque appel à la procédure de révision (Algorithm 2)$revise(c_{xy}, x)$, nous essayons de trouver un support pour chaque valeur $v \in d_x$ dans $d_y$. Plus précisément, nous recherchons d'abord le premier support $s_{min}$ (line 3) ; si aucun support n'est trouvé la valeur $v$ est supprimée du domaine de $x$, sinon nous cherchons aussi le dernier support $s_{max}$ (line 7). De cette manière nous obtenons à la fin de l'algorithme d'IAC un intervalle $I_{(x,v),y} = [s_{min}, s_{max}]$ contenant tous les supports de $v$ dans $y$. Quand la valeur $v$ admet seulement un support, l'intervalle est dans ce cas réduit à l'unique valeur supportée. Comme nous pouvons le voir dans la suite, tous ces intervalles de supports calculés mènent à une réduction significative du nombre de vérification de support durant les étapes de prétraitement et de recherche (section 4).

**Algorithm 1**: AC($\mathcal{P}$) : boolean

```
1  for each x ∈ X do
2      for each y ∈ X | y ≠ x, c_xy ∈ C do
3          for each v ∈ D_x do
4              I_(x,v),y ← [v_1, v_|D_y|] | D_y = {v_1, ..., v_|D_y|}
5  Q ← {(c_xy, x) | c_xy ∈ C}
6  while Q ≠ ∅ do
7      pick and delete (c_xy, x) from Q
8      if revise(c_xy, x) then
9          if D_x = ∅ then
10             return false
11         Q ← Q ∪ {(c_xz, z) | z ≠ y}
12 return true
```

**Algorithm 2**: revise($c_{xy} \in \mathcal{C}, x \in \mathcal{X}$) : boolean

```
1  removed ← false
2  for each v ∈ D_x do
3      if not(findFirst(c_xy, x, v)) then
4          removeValue(x, v)
5          removed ← true
6      else
7          findLast(c_xy, x, v)
8  return removed
```

La seconde partie de notre prétraitement, appelé IPC, exploite les intervalles de supports calculés grâce à IAC. Le but est d'appliquer un filtrage partiel de consistance de chemin. Ici, nous considérons seulement la forme conservative de consistance de chemin, i.e., le graphe de contraintes n'est pas modifié. Nous appliquons seulement la consistance de chemin sur les contraintes existantes. L'intégration des intervalles de supports avec cette forme partielle est décrite dans la définition suivante.

**Définition 5.** *Soit* $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ *un CSP binaire.* $\mathcal{P}$ *est Intervalle Chemin Consistant conservatif (CIPC) ssi* $\forall x, y, z \in \mathcal{X} \mid \exists c_{xy}, c_{xz}, c_{yz} \in \mathcal{C}, \forall a \in d_x, \forall b \in d_y$ *tel que* $(a, b) \in R_{c_{xy}}$, $I_{(x,a),z} \bigcap I_{(y,b),z} \neq \emptyset$.

Notons que si l'intersection est vide alors le tuple $(a, b)$ peut être supprimé de la relation $R_{c_{xy}}$. Dans la suite, pour simplifier les notations, nous utiliserons $I_{(x,a),y,z}$ pour désigner l'union ( $\bigcup_{b \in D_y \ / \ (a,b) \in R_{c_{xy}}} I_{(y,b),z}$).

Nous mentionnons que $I_{(x,a),y,z}$ ne couvre pas forcément la totalité de l'intervalle $I_{(x,a),z}$. Par conséquent, l'intervalle $I_{(x,a),z}$ peut être réduit sans effectuer de tests de consistance supplémentaires, et ceci grâce à une simple opération d'intersection. Nous obtenons donc un intervalle de supports plus petit pour $(x, a)$. La propriété suivante montre, formellement, comment les intervalles de supports peuvent être réduits de cette façon :

**Propriété 1.** *Soit* $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ *un CSP binaire.* $\forall x, y, z \in \mathcal{X} \mid \exists c_{xy}, c_{xz}, c_{yz} \in \mathcal{C}, \forall a \in d_x$, *nous avons* $I_{(x,a),z} = I_{(x,a),z} \cap I_{(x,a),y,z}$.

*Démonstration.* Soit $a \in d_x$, les valeurs dans $I_{(x,a),z}$ n'appartenant pas à $I_{(x,a),y,z}$ n'ont pas de supports dans $D_y$. Donc, ces valeurs ne doivent pas être dans $I_{(x,a),z}$. □

La figure 1, présente un CSP de 3 variables $x$, $y$ et $z$ où :
- $D_x = \{a, b\}$ , $D_y = \{a, b, c\}$ et $D_z = \{1, 2, \ldots, 8\}$
- $R_{c_{xy}} = \{(a, a), (a, b), (a, c), (b, b)\}$
- $I_{(x,a),z} = [1, 5]$ et $I_{(x,b),z} = [5, 8]$
- $I_{(y,a),z} = [2, 4]$, $I_{(y,b),z} = [3, 6]$ et $I_{(y,c),z} = [7, 8]$

Dans cette figure, les lignes pleines indiquent la nouvelle situation après l'application de IAC et CIPC. Notons que nous utilisons la variable $z$ par rapport à $x$ et $y$ pour montrer comment nous gérons les intervalles de supports. L'effet de l'application de IAC et CIPC concerne la contrainte $c_{xy}$ et les domaines de $x$ et $y$. Le tuple $(a, c) \in R_{c_{xy}}$ est supprimé (voir la définition 5). Par conséquent, la valeur $c \in D_y$ sera supprimée par AC. De plus, les intervalles de supports pour les valeurs de $D_x$ sont réduits grâce à la propriété 1. Par exemple, nous avions initialement $I_{(x,a),y,z} = [2, 8]$ et $I_{(x,a),z} = [1, 5]$. Après l'application de CIPC, nous obtenons $I_{(x,a),z} = [2, 5]$.

L'algorithme 3 montre le processus de propagation utilisant les intervalles de supports. La fonction *propagate* supprime les tuples qui ne vérifie pas IPC (ligne 5) et réduit les

intervalles de supports (ligne 8). En plus, certaines valeurs peuvent être supprimées sans tests de consistance supplémentaires (lignes 9 et 10).

**Algorithm 3**: propagate($x, y, z \in \mathcal{X}$) : boolean

```
1  for each v1 ∈ D_x do
2      I_(x,v1),y,z ← ∅
3      for each v2 ∈ D_y | (v1, v2) ∈ R_cxy do
4          if I_(y,v2),z) ∩ I_(x,v1),z = ∅ then
5              deleteTuple(c_xy, (v1, v2))
6          else
7              I_(x,v1),y,z ← I_(x,v1),y,z ∪ I_(y,v2),z
8      I_(x,v1),z ← I_(x,v1),z ∩ I_(x,v1),y,z
9      if I_(x,v1),z = ∅ then
10         removeValue(x, v1)
11 return D_x ≠ ∅
```

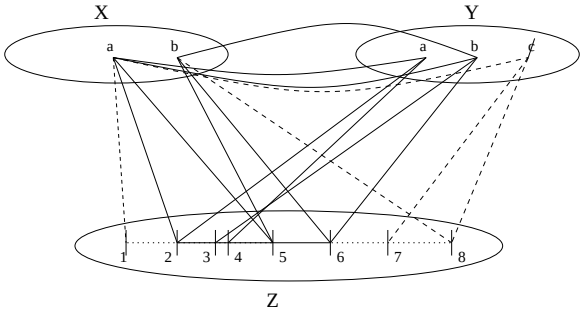


FIG. 1 – Le graphe après l'application de IPC

Nous rappelons que, après l'application de la fonction CIPC, des valeurs peuvent perdre leurs supports, i.e. elles deviennent arc inconsistantes. Nous devons donc appliquer une fois la procédure IAC sur le CSP et lancer de nouveau CIPC jusqu'à obtenir un point fixe. Cependant, pour des raisons de complexité en pratique, nous limitons l'application de CIPC à une passe. Si nous reprenons l'exemple de la figure 1, nous remarquons que le tuple $(a, c)$ ne vérifie pas la CIPC, donc il doit être supprimé. Par conséquent, la valeur $c \in D_y$ devient arc inconsistante et nous devons la supprimer de son domaine.

## 4 Intégration partielle de PC pendant la recherche

Dans cette section, nous montrons comment l'utilisation des intervalles de supports peut éviter d'explorer, pendant la résolution, certaines parties de l'espace de recherche. En effet, à chaque étape de la recherche, l'affectation d'une variable décrémente le degré de ses voisines et certaines d'entre elles peuvent devenir des variables singletons (des variables de degré 1) ou doubletons (degré 2). Pour ce type de variables nous n'avons pas besoin de continuer le processus de recherche classique et nous pouvons les déconnecter sous certaines conditions. Les variables singletons peuvent être déconnectées si le CSP vérifie AC ce qui est le cas puisque nous utilisons l'algorithme MAC [13]. Quant aux variables de degré 2, elles seront déconnectées si elles vérifient DPC par rapport à ses 2 voisines. Par exemple, la variable $z$ de la figure 1 peut être déconnectée après l'application de DPC sur ses voisines ($x$ et $y$). Dans [3], les auteurs ont proposé une manière similaire pour déconnecter les variables doubletons mais sans les intervalles de supports et sans fournir de preuves pratiques sur son utilité.

Ici, quand nous appliquons PC sur un couple de variables $(x, y)$ par rapport à une autre variable $z$, nous ne parcourons pas l'ensemble du domaine de $z$ mais seulement un sous-ensemble de celui-ci. Par exemple, si nous cherchons un support pour $(v_1, v_2) \in R_{c_{xy}}$ dans $D_z$ nous vérifions uniquement les valeurs dans l'intersection entre les intervalles de supports $I_{(x,v_1),z}$ et $I_{(y,v_2),z}$ (ligne 4 de l'algorithme 4). Notons que, dans le cas d'une intersection vide, nous pouvons immédiatement supprimer le tuple $(v_1, v_2)$ sans aucun test de consistance. Après l'application de l'algorithme $propagate2$, les variables doubletons peuvent être déconnectées, et ensuite nous choisissons une nouvelle variable à assigner. L'algorithme $propagate2$ applique DPC basé sur les intervalles de supports.

**Algorithm 4**: propagate2($c_{xy} \in \mathcal{C}, x \in \mathcal{X}$) : boolean

```
1  for each v1 ∈ D_x do
2      for each v2 ∈ D_y|(v1, v2) ∈ R_cxy do
3          support ← false
4          for each v3 ∈ I_(y,v2),z ∩ I_(x,v1),z do
5              if (v1, v3) ∈ R_cxz and (v2, v3) ∈ R_cyz then
6                  support ← true
7                  break
8          if not support then
9              deleteTuple(c_xy, (v1, v2))
10     return R_cxy ≠ ∅
```

## 5 Résultats préliminaires

Pour évaluer l'efficacité en pratique de l'application des intervalles de supports, nous avons effectué des expérimentations sur des instances binaires de la deuxième compétition internationale de CSP (http ://www.cril.univ-artois.fr/CPAI06/). Dans nos expérimentations, nous considérons l'algorithme MAC[1] et nous le comparons avec différentes implémentations basées sur la notion d'intervalles de supports. Notre objectif est d'étudier l'impact de l'application du prétraitement des intervalles d'une part (algorithme *MacP*) et l'effet de la déconnexion des variables

[1] Dans MAC, nous utilisons AC8[2] comme algorithme de consistance d'arc

sous certaines conditions (voir la section 4) d'autre part (algorithme *Mac+* [3]). De plus, nous évaluons la performance de notre méthode en utilisant à la fois le prétraitement et la déconnexion de variables pendant la recherche (algorithme *MacP+*).

Nous présentons, dans le tableau 1, un ensemble de problèmes représentatifs. Ce tableau contient des classes aléatoires (ehi-85, qcp-20 et tigntness0.*), des instances académiques (d(omino)-800-800, k(nights)-25-9, q(ueens)K(nights)-15-5-add) et des problèmes industriels (fapp06-0500-6, graphmod-9-f10, scen11).

| instance | MAC | Mac+ | MacP | MacP+ |
|---|---|---|---|---|
| d-800-800 | - | - | 107.03 | 107.05 |
| ehi-85 | 1.48 | 1.5 | 1.49 | 1.5 |
| fapp06-0500-6 | 128.86 | 137.84 | - | - |
| k-25-9 | 21.08 | 22.26 | 14.29 | 14.2 |
| qcp-20 | 466.3 | 464.69 | 92.89 | 94.71 |
| qK-15-5-add | 14.57 | 23.49 | 8.26 | 8.61 |
| graphMod-9-f10 | 407.23 | 425.4 | 11.51 | 11.1 |
| scen11-f8 | 306.01 | 306.66 | 118.54 | 117.58 |
| tightness0.35 | 15.88 | 16.51 | 17.64 | 17.7 |
| tightness0.9 | 271.54 | 123.09 | 261.07 | 111.28 |

TAB. 1 – Résultats avec $\frac{dom}{wdeg}$

Pour les classes "tightness0.*", notre méthode semble être plus efficace que MAC lorsque la dureté des contraintes augmente, particulièrement quand nous exploitons la déconnexion. Cela est dû au fait que les problèmes aléatoires avec des contraintes dures ont un réseau de contrainte peu dense. Par conséquent, nous pouvons déconnecter plus de variables. Concernant les problèmes académiques, nous remarquons que, en général, nous améliorons l'algorithme MAC. Nous pouvons remarquer que MAC et Mac+ ne sont pas capables de résoudre le problème d-800-800 dans le temps imparti (1200 sec.). Pour les problèmes industriels, l'exploitation des intervalles de supports améliore l'algorithme MAC. En ce qui concerne l'instance fapp06-500-6, l'échec de notre méthode peut être expliqué par le fait que les domaines de cette instance sont grands et le réseau de contraintes est dense ce qui signifie que l'étape de prétraitement peut être coûteux en temps.

## 6 Conclusion

Dans ce papier, nous avons présenté un prétraitement permettant de calculer les intervalles de supports dans les CSPs. Il exploite la propriété de consistance d'arc basée sur les intervalles de supports. Il permet de réduire le nombre de tests de consistance lorsque l'on applique AC pendant la recherche. Nous avons montré que la consistance de chemin conservative peut être appliquée sans effectuer de tests de consistance supplémentaires grâce l'utilisation des intervalles de supports. Les résultats préliminaires sont prometteurs et ils nous encouragent à poursuivre ce travail et à étudier cette piste plus en profondeur.

## Références


[1] Christian Bessière and Marie-Odile Cordier. Arc-consistency and arc-consitency again. In *In proceedings of AAAI*, Washington D.C., 1993.

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

[3] A. Chmeiss and L. Sais. About the use of local consistency in solving csps. In *Proceedings of 12th International Conference on Tools with Artificial Intelligence*, pages 104–107, Vancouver, Canada, 2000.

[4] M.C. Cooper, D.A. Cohen, and P.G. Jeavons. Characterizing tractable constraints. *Arti. Intell.*, 65 :347–361, 1994.

[5] R. Debruyne. A strong local consistency for constraint satisfaction. In *Proceedings of 11th International Conference on Tools with Artificial Intelligence*, pages 202–209, Chicago, Ill, 1999.

[6] R. Dechter. Enhancement schemes for constraint-satisfaction problems : Backjumping, learning and cutset decomposition. *Arti. Intell.*, 41 :273–312, 1990.

[7] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34 :1–38, 1988.

[8] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38 :353–366, 1989.

[9] P. Jégou. Decomposition of domains based on the structure of finite constraints satisfaction problems. In *AAAI'93*, pages 731–736, 1993.

[10] C. Lecoutre and J. Vion. Bound consistencies for the csp. In *Proceeding of the second international workshop "Constraint Propagation And Implementation (CPAI'2005)" held with the 10th International Conference on Principles and Practice of Constraint Programming (CP'2005)*, Sitges, Spain, September 2005.

[11] U. Montanari. Networks of constraints : fundamental properties and applications to picture processing. *Inform. Sci*, 7 :95–132, 1974.

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

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