Consistance duale conservative

Christophe Lecoutre, Stéphane Cardon, Julien Vion

Consistance duale conservative

Christophe Lecoutre **Stéphane Cardon** **Julien Vion**

CRIL - CNRS FRE 2499
Université d'artois
rue de l'université, SP 16
62307 Lens cedex, France

{lecoutre,cardon,vion}@cril.univ-artois.fr

Résumé

Les consistances sont des propriétés de réseaux de contraintes qui peuvent être exploitées afin de générer des inférences. Lorsqu'un nombre important d'inférences peut être effectué, il devient alors plus facile de résoudre les réseaux à l'aide par exemple d'une recherche systématique. Dans ce papier, nous nous intéressons aux consistances de relation, i.e. aux consistances qui permettent d'identifier des couples de valeurs inconsistantes. Nous proposons une nouvelle consistance, appelée consistance duale (DC pour Dual Consistency), et nous la comparons à la consistance de chemin (PC pour Path Consistency). Nous montrons que la DC conservative (CDC), i.e. DC telle que seules les relations associées aux contraintes du réseau soient filtrées, est plus forte, en terme de filtrage, que la PC conservative (CPC). En suivant l'approche de Mac Gregor, nous introduisons un algorithme pour établir la CDC (forte) avec une complexité spatiale très faible dans le pire des cas. Bien que l'efficacité relative de l'algorithme introduit dépend en partie de la densité du graphe de contraintes, l'expérimentation que nous avons conduite montre que, sur de nombreuses séries d'instances, établir CDC à l'aide de cet algorithme est largement plus rapide qu'établir CPC à l'aide d'algorithmes directement adaptés de PC-8 ou PC-2001. Par ailleurs, nous avons observé qu'établir CDC lors d'une phase de pré-traitement permet une accélération significative de la résolution de certaines instances structurées difficiles.

Abstract

Consistencies are properties of Constraint Networks (CNs) that can be exploited in order to make inferences. When a significant amount of such inferences can be performed, CNs are much easier to solve. In this paper, we interest ourselves in relation filtering consistencies, i.e. consistencies that allow to identify inconsistent pairs of values. We propose a new consistency called Dual Consistency (DC) and relate it to Path Consistency (PC). We show that Conservative DC (CDC, i.e. DC with only relations associated with the constraints of the network considered) is more powerful, in terms of filtering, than Conservative PC (CPC). Following the approach of Mac Gregor, we introduce an algorithm to establish (strong) CDC with a very low worst-case space complexity. Even if the relative efficiency of the algorithm introduced to establish (strong) CDC partly depends on the density of the constraint graph, the experiments we have conducted show that, on many series of CSP instances, CDC is largely faster than CPC (up to more than one order of magnitude). Besides, we have observed that enforcing CDC in a preprocessing stage can significantly speed up the resolution of hard structured instances.

1 Introduction

Les consistances sont des propriétés de réseaux de contraintes (CN pour Constraint Network) qui peuvent être exploitées afin de générer des inférences. Ces dernières permettent de simplifier le problème, c'est à dire de réduire l'espace de recherche de telle sorte que le réseau soit plus simple à résoudre (i.e. trouver une solution ou prouver qu'aucune solution n'existe). La plupart des solveurs intègre recherche et inférence.

Aujourd'hui, les consistances les plus abouties sont les consistances de domaine (domain filtering consistencies) [6]. Elles permettent d'identifier des valeurs inconsistantes qui peuvent être éliminées du domaine des variables correspondantes. Pour les contraintes binaires, on peut citer

la consistance d'arc (AC pour Arc Consistency) et la singleton consistance d'arc (SAC pour Singleton Arc Consistency) [2]. La consistance d'arc généralisée (GAC pour Generalized Arc Consistency) et l'inter-consistance inverse (PWIC pour Pairwise Inverse Consistency) [13] sont définies pour les contraintes non binaires. Une autre classe de consistances comprend celles qui permettent d'identifier des couples de valeurs inconsistantes. On peut les appeler consistances de relation (relation filtering consistencies) – mais on ne doit pas les confondre avec les consistances relationnelles (relational consistencies) [8]. La plus célèbre d'entre elles est la consistance de chemin (PC pour Path Consistency). Un couple de valeurs pour un couple de variables est chemin-consistant ssi il peut être étendu à une instantiation consistante pour toute autre variable. De nombreux algorithmes ont été proposés pour établir PC sur un réseau donné : PC4 [12], PC8 [4], PC2001 [3], etc.

La consistance de chemin, et plus généralement les consistances de relation, sont quelque peu négligées par les développeurs de systèmes de programmation par contraintes. La raison principale est qu'exploiter de telles consistances implique de modifier les relations associées aux contraintes, et plus ennuyeux, de modifier la structure du graphe de contraintes associé au réseau. En effet, lorsqu'un couple de valeurs est identifié comme étant chemin-inconsistant, il doit être éliminé du réseau. Lorsqu'il n'existe pas de contrainte dans le réseau portant sur les deux variables impliquées, une nouvelle contrainte doit être insérée (modifiant en conséquence le graphe de contraintes). Par exemple, le CN qui représente l'instance $scen$-11 du problème d'allocation de fréquences radio (RLFAP pour Radio-Frequency Assignment Problem) comporte 680 variables et 4103 contraintes. Dans le pire des cas, établir PC sur ce réseau entraînera la création de $C^2_{680} - 4103 = 226757$ nouvelles contraintes, ce qui peut réellement être contre-productif aussi bien en temps qu'en espace.

Cependant, il est possible d'éviter le défaut principal de PC en adoptant une approche *conservative*. cela signifie simplement que nous pouvons limiter notre attention aux contraintes existantes lorsque des couples de valeurs inconsistantes sont recherchées. On obtient alors la consistance de chemin conservative (CPC pour Conservative Path Consistency) [5]. Nous définissons une nouvelle consistance de relation, appelée consistance duale (DC pour Dual Consistency), qui exploite le résultat de tests singletons effectués avec AC. Nous montrons que PC est plus forte (en terme de filtrage) que DC conservative (CDC) qui elle-même est plus forte que CPC – CDC peut filtrer plus de couples de valeurs inconsistantes que CPC. Nous proposons un algorithme simple qui établit la CDC (forte), et qui peut être considérée comme une adaptation de celui proposé par [11] pour établir la PC forte. Cet algorithme ne nécessite aucune structure de données spécifique (hormis celles de l'algorithme AC sous-jacent) et peut converger rapidement vers un point fixe (unique).

2 Consistances

Un réseau de contraintes (CN pour Constraint Network) P est un couple $(\mathscr{X}, \mathscr{C})$ où $\mathscr{X}$ est un ensemble fini de n variables et $\mathscr{C}$ un ensemble fini de e contraintes. Chaque variable $X \in \mathscr{X}$ possède un domaine, noté $dom^P(X)$, qui représente l'ensemble des valeurs autorisées pour X. Chaque contrainte $C \in \mathscr{C}$ implique un sous-ensemble de variables de $\mathscr{X}$, appelé portée et noté $scp(C)$, et est définie par une relation $rel^P(C)$, qui représente l'ensemble des tuples autorisés pour les variables de sa portée. Lorsque cela sera possible (i.e. sans ambiguïté), on notera $dom(X)$ et $rel(C)$ au lieu de $dom^P(X)$ et $rel^P(C)$. X_a représente un couple (X, a) avec $X \in \mathscr{X}$ et $a \in dom(X)$, et nous dirons que X_a est une valeur de P. λ représente le nombre de tuples autorisés sur l'ensemble des contraintes de P (avec $\lambda = \sum_{C \in \mathscr{C}} |rel(C)|$), et K représente le nombre de 3-cliques dans le graphe de contraintes associé à P. Si P et Q sont deux CNs définis sur les mêmes ensembles de variables $\mathscr{X}$ et de contraintes $\mathscr{C}$, alors nous écrirons $P \leq Q$ ssi $\forall X \in \mathscr{X}$, $dom^P(X) \subseteq dom^Q(X)$ and $\forall C \in \mathscr{C}\ rel^P(C) \subseteq rel^Q(C)$. $P < Q$ ssi $P \leq Q$ and $\exists X \in \mathscr{X} \mid dom^P(X) \subset dom^Q(X)$ ou $\exists C \in \mathscr{C} \mid rel^P(C) \subset rel^Q(C)$. Une contrainte binaire est une contrainte qui porte uniquement sur deux variables. Dans la suite, nous restreindrons notre attention aux réseaux binaires, i.e., aux réseaux qui impliquent uniquement des contraintes binaires. En outre, sans perte de généralité, nous considérerons que les contraintes sont normalisées, c'est-à-dire que la même portée ne peut pas être partagée par des contraintes distinctes [1]. La densité D d'un réseau binaire est définie par le rapport e/C^2_n.

Une solution d'un réseau de contraintes est une assignation de valeurs à l'ensemble des variables telle que toutes les contraintes soient satisfaites. Un réseau de contraintes est satisfiable ssi il admet au moins une solution. Le problème de satisfaction de contraintes (CSP pour Constraint Satisfaction Problem), qui consiste à déterminer si un réseau de contraintes donné est satisfiable, est NP-complet. Une instance CSP est alors définie par un réseau de contraintes et le résoudre consiste à trouver une (ou plusieurs) solution(s) ou alors à prouver son insatisfiabilité. Pour résoudre une instance CSP, on peut modifier le réseau de contraintes en utilisant des méthodes d'inférence ou de recherche [7]. L'inférence consiste à simplifier le réseau de manière à ce qu'il soit plus simple à résoudre. Une approche classique pour l'inférence est l'exploitation de certaines propriétés appelées consistances. Il s'agit alors d'établir telle ou telle consistance sur un réseau tout en préservant l'équivalence, c'est-à-dire l'ensemble des solutions. Cela revient à identifier et éliminer des valeurs in-

consistantes (e.g. avec la consistance), des couples inconsistants de valeurs (e.g. avec la consistance de chemin), etc. Inconsistant signifie que les valeurs, les couples de valeurs, etc. identifiés comme tels correspondent à des nogoods, c'est-à-dire, ne peuvent participer à aucune solution. Nous commençons par introduire les consistances au niveau des couples de valeurs. A partir de maintenant, nous considérons donné un réseau de contraintes $P = (\mathscr{X}, \mathscr{C})$.

Definition 1. Un couple de valeurs (X_a, Y_b) du réseau P tel que $X \neq Y$ est :
- arc-consistant (AC) si et seulement si $\nexists C \in \mathscr{C} \mid scp(C) = \{X, Y\}$ ou $(X_a, Y_b) \in rel(C)$.
- chemin-consistant (PC) si et seulement si (X_a, Y_b) est AC et $\forall Z \in \mathscr{X} | Z \neq X \wedge Z \neq Y$, $\exists c \in dom(Z)$ tel que (X_a, Z_c) est AC et (Y_b, Z_c) est AC.
- conservative chemin-consistant (CPC) si et seulement si $\nexists C \in \mathscr{C} \mid scp(C) = \{X, Y\}$ ou (X_a, Y_b) est PC.

Dès lors, nous pouvons étendre ces notions de consistances pour le réseau de contraintes.

Definition 2. Une valeur X_a de P est AC si et seulement si $\forall Y (\neq X) \in \mathscr{X}$, $\exists b \in dom(Y) | (X_a, Y_b)$ est AC. P est AC si et seulement si $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ et $\forall a \in dom(X)$, X_a est AC.

Definition 3. Une couple de variables (X, Y) distinctes de $\mathscr{X}$, est PC (resp. CPC) si et seulement si $\forall a \in dom(X)$, $\forall b \in dom(Y)$, (X_a, Y_b) est PC (resp. CPC). P est PC (resp. CPC) si et seulement tout couple de variables distinctes de $\mathscr{X}$ est PC (resp. CPC).

Toutes les consistances ϕ mises en avant dans ce papier admettent la propriété (de confluence) suivante : pour tout réseau P, le plus grand sous-réseau de P qui est ϕ-cohérent existe. Il est noté $\phi(P)$ et il est possible de le déterminer en temps polynômial. Par exemple, $AC(P)$ représente le réseau de contraintes obtenu après avoir applique la consistance d'arc sur le réseau P. $AC(P)$ est tel que toutes valeurs de P qui sont arc incohérentes sont supprimées. Remarquons que s'il existe une variable de $AC(P)$ ayant un domaine vide, alors le réseau P n'admet pas de solution. Nous notons cela $AC(P) = \bot$. Pour une quelconque valeur X_a, nous noterons $X_a \in AC(P)$ si et seulement si $a \in dom^{AC(P)}(X)$ (on considérera que $X_a \notin \bot$). Finalement, $P|_{X=a}$ représente le réseau de contraintes obtenu à partir de P en réduisant le domaine de X à l'unique valeur $\{a\}$.

Intéressons nous maintenant à la singleton consistance d'arc (SAC) et à une nouvelle consistance, appelée consistance duale (DC).

Definition 4. Une valeur X_a de P est SAC si et seulement si $AC(P|_{X=a}) \neq \bot$. P est SAC si et seulement si $\forall X \in \mathscr{X}$, $dom(X) \neq \emptyset$ et $\forall a \in dom(X)$, X_a est SAC.

Definition 5. Un couple de valeurs (X_a, Y_b) de P tel que $X \neq Y$ est :
- dual-consistant (DC) si et seulement si $Y_b \in AC(P_{|X=a})$ et $X_a \in AC(P_{|Y=b})$.
- conservative dual-consistant (CDC) si et seulement si $\nexists C \in \mathscr{C} \mid scp(C) = \{X, Y\}$ ou (X_a, Y_b) est DC.

Definition 6. Un couple de variables (X, Y) distinctes appartenant à $\mathscr{X}$, est DC (resp. CDC) si et seulement si $\forall a \in dom(X)$, $\forall b \in dom(Y)$, (X_a, Y_b) est DC (resp. CDC). P est DC (resp. CDC) si et seulement si tout couple de variables distinctes de $\mathscr{X}$ est DC (resp. CDC).

Depuis une quelconque consistance de relation, i.e. une consistance filtrant les relations d'un réseau de contraintes, il est possible d'obtenir une nouvelle consistance en y greffant la consistance d'arc. D'un point de vue classique, un réseau est dit fortement chemin-consistant, noté sPC, si et seulement si il est à la fois chemin-consistant et arc-consistant. D'une manière similaire, nous définissons sCPC, sDC et sCDC.

3 Étude qualitative

Dans le but de pouvoir comparer l'efficacité des différentes consistances présentées précédemment, nous avons besoin d'introduire une relation de pré-ordre comme dans [6]. Une consistance locale ϕ est plus forte qu'une autre consistance locale ψ, noté $\phi \succeq \psi$, si et seulement si pour tout réseau de contraintes P, si P est ϕ-consistant alors il est ψ-consistant. ϕ est strictement plus forte que ψ, noté $\phi \succ \psi$, si et seulement si $\phi \succeq \psi$ et il existe au moins un réseau de contraintes P qui est ϕ-consistant mais pas ψ-consistant. Pour tout couple de consistances (ϕ, ψ) présentées dans ce papier, nous savons que $\phi \succeq \psi$ si et seulement si pour tout réseau P, $\phi(P) \geq \psi(P)$ et $\phi \succ \psi$ si et seulement si $\phi \succeq \psi$ et il existe un réseau P tel que $\phi(P) > \psi(P)$. De plus, toutes ces consistances sont monotones : pour tout couple de réseaux (P, Q) tel que $P \geq Q$, nous avons $\phi(P) \geq \phi(Q)$. Finalement, nous définissons $\phi \circ \psi(P)$ comme étant $\phi(\psi(P))$ et $(\phi \circ \psi)^{n+1}(P)$ comme $\phi \circ \psi \circ (\phi \circ \psi)^n(P)$.

Il n'est pas trop surprenant de constater que DC et PC apparaissent comme des consistances équivalentes. En effet, McGregor a proposé un algorithme basé sur AC pour établir PC [11]. Nous allons en proposer une démonstration dans notre contexte :

Proposition 1. *DC = PC*

Preuve. Nous allons montrer que pour tout réseau P, $DC(P) = PC(P)$. Soit (X_a, Y_b) un couple de valeurs de P.
- Si (X_a, Y_b) n'est pas PC, soit (X_a, Y_b) n'est pas AC et alors (X_a, Y_b) n'est pas DC, soit $\exists Z \in \mathscr{X} \mid \forall c \in dom(Z)$, (X_a, Z_c) n'est pas AC ou (Y_b, Z_c)

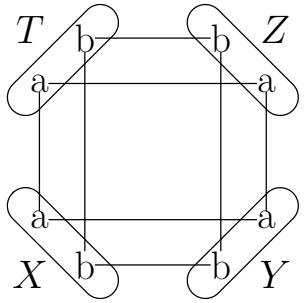

FIG. 1 – Un réseau (il n'y pas de contraintes reliant X avec Z et Y avec T) qui est CDC et sCDC mais ni sPC, ni PC. Par exemple, (X_a, Z_b) n'est pas PC.

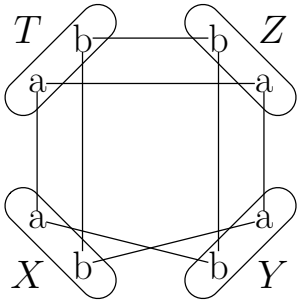

FIG. 2 – Un réseau (il n'y a pas de contraintes reliant X avec Z et Y avec T) qui est CPC et sCPC mais ni sCDC, ni CDC. Par exemple, (X_a, T_a) n'est pas CDC puisque $AC(P_{|X=a}) = \bot$.

n'est pas AC. Dans ce cas, nous savons que $Y_b \notin AC(P_{|X=a})$ puisqu'après avoir filtré le réseau $P_{|X=a}$ par la consistance d'arc, toute valeur c de $dom(Z)$ est telle que (X_a, Z_c) est AC. Nécessairement, par hypothèse, toutes ces valeurs sont incompatibles avec Y_b, ce qui implique que b est supprimée de $dom(Y)$ après avoir établi AC. Donc, (X_a, Y_b) n'est pas DC.

– Si (X_a, Y_b) n'est pas DC, alors cela veut dire que $Y_b \notin AC(P_{|X=a})$ (ou $X_a \notin AC(P_{|Y=b})$). Considérons l'hypothèse de récurrence H(n) suivante : si le nombre de révisions (c'est-à-dire le nombre d'étapes réalisées par un algorithme tel AC2001) pour supprimer b de $dom(Y)$ après avoir établi AC sur $P_{|X=a}$ est plus petit ou égal à n alors le couple (X_a, Y_b) n'appartient pas à $PC(P)$. H(1) est vérifiée puisque dans notre cas cela veut dire que (X_a, Y_b) n'est pas AC. Supposons que H(n) est vraie et démontrons que H($n+1$) est vérifiée. Si b est supprimé de $dom(Y)$ après $n+1$ révisions d'un filtrage AC sur $P_{|X=a}$, alors la dernière révision concerne une contrainte reliée à Y et une autre variable Z. Quelle que soit c de $dom(Z)$, qui est un support pour Y_b, c a été supprimée après au plus n révisions. Par hypothèse, on en déduit que pour ces valeurs c, (X_a, Z_c) n'appartient pas à $PC(P)$. De ce fait nous pouvons conclure que (X_a, Y_b) n'est pas PC.

□

Par la suite, nous allons étudier les relations existantes entre la consistance duale conservative (CDC) et la consistance de chemin et sa variante conservative.

Proposition 2. *PC ≻ CDC.*

Preuve. Clairement, DC ⪰ CDC. Donc, par la proposition 1, nous obtenons PC ⪰ CDC. De plus, la figure 1 met en avant un réseau (les arrêtes représentent les tuples autorisés) qui est CDC mais pas PC. □

Proposition 3. *CDC ≻ CPC.*

Preuve. Par un raisonnement similaire à la première partie de la démonstration de la proposition 1, nous pouvons montrer que CDC ⪰ CPC. De plus, la figure 2 met en avant un réseau qui est CPC mais pas CDC. En effet, comme il n'y a pas de 3-clique, le réseau est obligatoirement CPC. □

Avant d'entrer dans les détails des relations entre sPC, sCDC et sCPC, remarquons qu'établir AC (une seule fois) sur un réseau PC est suffisant pour obtenir un réseau sPC. Ce fait bien connu reste vrai pour CDC. En effet, toute valeur arc-inconsistante est complètement isolée, comme montré dans la proposition suivante (pour laquelle la démonstration est omise).

Proposition 4. *Soit un réseau de contraintes $P = (\mathscr{X}, \mathscr{C})$ qui est PC (resp. CDC). Une valeur X_a de P n'est pas AC si et seulement si $\forall Y \in \mathscr{X}$ (resp. t.q. $\exists C \in \mathscr{C} \mid scp(C) = \{X, Y\}$), $\forall b \in dom(Y)$, (X_a, Y_b) n'est pas PC (resp. CDC).*

Corollary 1. *Soit un réseau de contraintes P. $AC \circ PC(P) = sPC(P)$ et $AC \circ CDC(P) = sCDC(P)$.*

Fait intéressant, la proposition précédente reste vérifiée pour CPC. Considérons l'exemple mis en avant par la figure 3. Supposons que Z_b soit supprimée. Si nous filtrons par CPC, tous les tuples en pointillés sont supprimés. Ensuite, si nous filtrons par AC, la valeur X_b est supprimée. Nous pouvons dès lors imaginer le même motif ($X' := X, Y' := Y, Z' := X, T := T'$) raccroché à notre figure par $X = Z'$. Et ainsi de suite, pour n'importe quel entier n, nous pouvons construire un réseau P tel que $(AC \circ CPC)^{n+1}(P) = sCPC(P)$ alors que $(AC \circ CPC)^n(P) \neq sCPC(P)$.

Proposition 5. *sPC ≻ sCDC.*

Preuve. Pour tout réseau P, $PC(P) \geq CDC(P)$ par la proposition 2. De plus, par monotonie, nous obtenons $AC \circ$

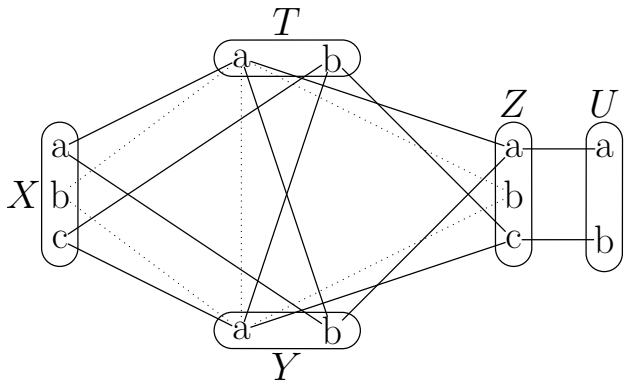

FIG. 3 – Motif montrant que $AC \circ CPC \neq sCPC$. Il n'y a pas de contrainte reliant X à Z.

$PC(P) \geq AC \circ CDC(P)$. Ainsi, par le corollaire 1, nous obtenons $sPC(P) \geq sCDC(P)$. Finalement, la figure 1 met en avant un réseau qui est sCDC mais pas sPC. □

Proposition 6. *$sCDC \succ sCPC$.*

Preuve. Considérons l'hypothèse de récurrence $H(n)$ suivante : pour tout réseau P, $AC \circ CDC(P) \geq (AC \circ CPC)^n(P)$. Pour $n = 1$, elle est immédiatement vérifiée puisque $CDC \succ CPC$. Maintenant, supposons $H(n)$ et montrons que $H(n+1)$ reste vraie. Nous avons, par hypothèse, $AC \circ CDC(P) \geq (AC \circ CPC)^n(P)$. Donc, nous obtenons $CPC \circ AC \circ CDC(P) \geq CPC \circ (AC \circ CPC)^n(P)$ par monotonie. Comme $AC \circ CDC(P) = sCDC(P)$, ce réseau est CDC et, par conséquent, CPC. De là, nous avons $CPC \circ AC \circ CDC(P) = AC \circ CDC(P) \geq CPC \circ (AC \circ CPC)^n(P)$. Et, par monotonie, nous avons $AC \circ AC \circ CDC(P) \geq AC \circ CPC \circ (AC \circ CPC)^n(P)$. Nous pouvons donc déduire $AC \circ CDC(P) \geq (AC \circ CPC)^{n+1}(P)$. Comme $sCPC = (AC \circ CPC)^n(P)$ pour un quelconque entier n fini, nous avons montré que $sCDC \succeq sCPC$. Finalement, la figure 2 met en avant un réseau qui est sCPC mais pas sCDC. □

La proposition suivante est triviale puisque toute valeur non SAC est supprimée par un algorithme de filtrage sCDC.

Proposition 7. *$sCDC \succ SAC$.*

4 Algorithme sCDC-1

L'algorithme que nous proposons pour établir sCDC est appelé sCDC-1. Il réalise des tests singletons successifs (rappelons qu'un test singleton est l'assignation d'une valeur à une variable suivie par l'application d'un opérateur d'inférence) jusqu'à ce qu'un point fixe soit atteint. La description est donnée dans le contexte de l'utilisation d'un

Algorithme 1 : sCDC-1($P = (\mathscr{X}, \mathscr{C})$: CN)

```
1 P ← AC(P, 𝒳)
2 X ← first(𝒳)
3 marker ← X
4 repeat
5     if check(P, X) then
6         P ← AC(P, {X})
7         marker ← X
8     X ← next-modulo(𝒳, X)
9 until X = marker
```

algorithme AC sous-jacent à gros grain, tel que AC3 [10], AC2001/3.1 [3] etr AC3^{rm} [9], avec un schéma de propagation orienté-variable. Si $P = (\mathscr{X}, \mathscr{C})$, alors $AC(P, Q)$ avec $Q \subseteq \mathscr{X}$ signifie que la consistance d'arc est établie sur P à partir de l'ensemble donné de propagation Q. Pour une description de AC, voir, par exemple, la fonction $propagateAC$ dans [2].

Pour établir sCDC sur un réseau donné P, la fonction $sCDC$-1 est appelée (voir algorithme 1). AC est établi tout d'abord (ligne 1), et ensuite, à chaque tour de la boucle principale, une variable différente est considérée : en supposant ici que $\mathscr{X}$ est ordonné, $first(\mathscr{X})$ retourne la première variable de $\mathscr{X}$ et $next\text{-}modulo(X, \mathscr{X})$ retourne la variable de $\mathscr{X}$ qui succède à X, ou $first(\mathscr{X})$ si X est la dernière variable dans l'ordre. L'appel à la fonction $check$ à la ligne 5 a pour but d'effectuer, si possible, des inférences à partir de X. Lorsque des inférences se produisent, $check$ retourne $true$ et la consistance d'arc est ré-établie (ligne 6). Pour gérer la terminaison, on utilise un marqueur initialisé avec la première variable de $\mathscr{X}$ (ligne 3) et mis à jour chaque fois que des inférences se produisent par rapport à la variable courante X (ligne 7).

Pour réaliser toutes les inférences sCDC par rapport à une variable X, la fonction $check$ est appelée (algorithme 2). Pour chaque valeur a du domaine de X, AC est établi sur $P|_{X=a}$. Si a est singleton arc-inconsistant, alors a est éliminé du domaine de X (ligne 5). Sinon (lignes 8 à 12), pour chaque valeur Y_b présent dans P et absent dans P' tel qu'il existe un contrainte portant sur X et Y, le tuple (X_a, Y_b) est éliminé de $rel(C)$.

Il y a une connexion forte entre l'algorithme sCDC-1 et l'algorithme proposé dans [11] pour établir sPC. L'algorithme que nous proposons peut être perçu comme un rafinnement de celui de Mac Gregor puisque nous pouvons établir CDC et avons intégré deux modifications importantes. Tout d'abord, AC est maintenu sur P durant l'exécution de l'algorithme de manière à pouvoir exécuter les tests singletons à partir d'un ensemble de propagation composé d'une seule variable (cela permet d'éviter de nombreuses révisions inutiles, en particulier lorsque le graphe de contraintes est clairsemé). Ensuite, la terminaison est gérée par un mécanisme amélioré. Il est possible de raisonner de la manière indiquée dans l'algorithme pour la termi-

Algorithme 2 : check(P : CN, X : Variable) : Boolean

```
1  modified ← false
2  foreach a ∈ dom^P(X) do
3      P' ← AC(P|_{X=a}, {X})
4      if P' = ⊥ then
5          remove a from dom^P(X)
6          modified ← true
7      else
8          foreach C ∈ 𝒞 | X ∈ scp(C) do
9              let Y be the second variable in scp(C)
10             foreach b ∈ dom^P(Y) | b ∉ dom^{P'}(Y) do
11                 remove (X_a, Y_b) from rel^P(C)
12                 modified ← true
13 return modified
```

naison, car pour toute variable X et tout couple de valeurs (a,b) de $dom(X)$, toute inférence concernant X_a (l'élimination de X_a ou l'élimination d'un tuple liant X_a à une autre valeur) n'a aucun impact sur $P_{X=b}$, et vice-versa. En pratique, ces deux améliorations contribuent à rendre l'algorithme plus performant.

Proposition 8. *L'algorithme sCDC-1 établit sCDC.*

Preuve. Tout d'abord, il est immédiat que toute inférence réalisée par sCDC-1 est correcte. La complétude est garantie par l'invariant suivant : lorsque $P' = AC(P|_{X=a}, \{X\})$ est exécuté à la ligne 3 de l'algorithme 2, nous avons $P' = AC(P|_{X=a}, \mathscr{X})$. La raison est que le réseau est maintenu arc-consistant chaque fois qu'une modification est effectuée (ligne 6 de l'algorithme 1) et que toute inférence réalisée par rapport à une valeur X_a n'a pas d'impact sur $P|_{X=b}$, où b représente toute autre valeur du domaine de la variable X. □

Proposition 9. *La complexité temporelle dans le pire des cas de sCDC-1 est $O(\lambda end^3)$ et la complexité spatiale dans le pire des cas est $O(ed^2)$.*

Preuve. Dans le pire des cas, la fonction $check$ peut être appelée λ fois pour une variable donnée, puisqu'entre deux appels successifs, au moins un tuple doit être éliminé d'une relation. Un algorithme AC optimal en $O(ed^2)$ tel que AC2001 peut être utilisé à la ligne 3, et éliminer des tuples inconsistants (lignes 8 à 12) est $O(nd)$. Comme on peut supposer $n < e$ (sinon, l'instance se compose de plusieurs sous-problèmes indépendants), un appel à $check$ est donc en $O(d(ed^2 + nd)) = O(ed^3)$. Ainsi, nous obtenons $O(\lambda end^3)$ pour sCDC-1. En terme d'espace, il est tout d'abord nécessaire de représenter domaines et relations : ceci est en $O(nd + ed^2) = O(ed^2)$. Les seules structures de données propres à sCDC-1 sont en fait celles de l'algorithme AC sous-jacent. Pour AC2001 ou AC3^{rm}, ceci est en $O(ed)$. La complexité spatiale dans le pire des cas est donc $O(ed^2)$. □

Comme λ est borné par $O(ed^2)$, sCDC-1 est également $O(e^2nd^5)$. Ceci peut sembler assez élevé, mais notre opinion est que sCDC-1 atteint rapidement un point fixe (i.e. le nombre de fois où la fonction $check$ est appelée pour une variable donnée est très faible) car les inférences portant sur les valeurs et les couples de valeurs inconsistants peuvent être pris immédiatement en compte. Le corollaire suivant indique également que le temps perdu à appliquer sCDC-1 sur un réseau déjà sCDC est tout à fait limité pourvu que la taille des domaines ne soit pas trop élevée.

Corollary 2. *Appliqué à un réseau sCDC, la complexité temporelle dans le pire des cas de sCDC-1 est $O(end^3)$.*

Il est également intéressant de comparer sCDC-1 avec les algorithmes établissant sCPC. Nous donnons quelques précisions sur les algorithmes sCPC8 et sCPC2001 qui peuvent être dérivés directement de PC8 [4] et PC2001 [3]. Tout d'abord, la complexité spatiale dans le pire des cas de ces deux algorithmes est respectivement $O(ed^2)$ et $O((e + K)d^2)$. La phase d'initialisation (nous ne discuterons pas de la phase de propagation) de ces deux algorithmes est la même : pour chaque 3-clique du (graphe de contraintes du) réseau, il faut vérifier que chaque tuple autorisé par une relation (associée à une contrainte) de cette clique est PC. Ceci est fait en $O(Kd^3)$. A partir de ces observations, il apparaît que, moins dense est le réseau, moins important est le nombre de 3-cliques, et plus faible est l'espace et le temps requis par ces algorithmes. D'un autre coté, si on considère des réseaux denses (e tendant vers C_n^2), on peut ajuster la complexité temporelle dans le pire des cas de la phase d'initialisation à $O(n^3d^3)$. Ceci constitue la même complexité temporelle dans le pire des cas qu'une seule passe (appeler $check$ pour toutes les variables une seul fois) de sCDC-1. Pour les réseaux peu denses, sCDC-1 devrait être donc plus lent que sCPC8 et sCPC2001, pour les réseaux denses (ou hautement structurés), nous pensons vraiment que sCDC-1, grâce à sa capacité d'effectuer des inférences rapidement, devrait être plus rapide.

L'algorithme que nous proposons est aussi en rapport avec les algorithmes SAC-OPT et SAC-SDS [2] proposés pour établir la singleton consistance d'arc. SAC-OPT admet une complexité temporelle dans le pire des cas en $O(end^3)$ mais une complexité spatiale élevée dans le pire des cas en $O(end^2)$. SAC-SDS relâche l'optimalité temporelle pour sauver de l'espace mémoire : sa complexité temporelle dans le pire des cas est en $O(end^4)$ et sa complexité spatiale dans le pire des cas est en $O(n^2d^2)$. sCDC-1 possède l'avantage de réduire plus efficacement l'espace de recherche, puisque $sCDC \succ SAC$, tout en limitant la complexité spatiale dans le pire des cas à $O(ed^2)$.

	AC3^{rm}	SAC-SDS	sCPC8 / sCPC2001	sCDC-1
Langford (4 instances)				
cpu	0.22	0.46	4.02 / 4.94	0.52
λ	105,854	105,769	75,727	75,727
blackhole-4-13 (7 instances) ($K = 92,769$; $D = 20\%$)				
cpu	1.26	19.39	140.54 / $-$	46.91
λ	8,206,320	8,206,320	8,206,320	7,702,906
$\langle 40, 180, 84, 0.9\rangle$ (20 instances) ($K = 12$; $D = 10\%$)				
cpu	0.71	10.57	2.28 / 2.02	17.42
λ	272,253	244,887	244,272	210,874
$\langle 40, 8, 753, 0.1\rangle$ (20 instances) ($K = 8,860$; $D = 96\%$)				
cpu	0.16	0.21	0.62 / 0.69	0.20
λ	43,320	43,320	43,318	43,318
job-shop enddr1 (10 instances) ($K = 600$; $D = 21\%$)				
cpu	1.58	4.06	7.91 / 10.54	4.67
λ	2,937,697	2,937,697	2,937,697	2,930,391
RLFAP scens (11 instances)				
cpu	0.86	$-$	25.96 / $-$	3.47
λ	1,674,286	$-$	1,471,132	1,469,286

TAB. 1 – Résultats expérimentaux (cpu en secondes)

5 Expérimentations

De façon à montrer l'intérêt pratique de l'approche décrite dans ce papier, nous avons mené une expérimentation intensive sur un PC équipé d'un processeur i686 2.4GHz avec 1024 MiB de RAM. Nous avons comparé le temps CPU et le niveau de filtrage (la valeur λ obtenue après avoir appliqué l'algorithme) de différents algorithmes appliqués simplement (i.e. sans recherche). Ces algorithmes sont une version optimisée de AC3^{rm} [9], SAC-SDS, sCPC8, sCPC2001 et sCDC-1. Plus précisément, nous avons utilisé AC $\circ$ CPC8 et AC $\circ$ CPC2001 comme approximation pour établir sCPC car nous avons observé qu'une seule passe était suffisant le plus souvent pour atteindre sCPC.

Nous avons tout d'abord testé les différents algorithmes sur différentes séries de problèmes[1]. Les contraintes définies en intention (i.e. par un prédicat) pour certaines instances ont été converties en extension (ceci n'eut d'impact significatif sur le temps cpu que pour les grosses instances des séries $fapp$). Comme attendu (voir table 1), sCDC-1 filtre plus que les autres algorithmes : plus petite est la valeur de λ, plus réduit est l'espace de recherche. Hormis la série $\langle 40, 180, 84, 0.9\rangle$, sCDC-1 est entre 2 et 8 fois plus rapide que sCPC8 et sCPC2001. En outre, sCDC-1 est presqu'aussi rapide que SAC-SDS qui, de son côté, nécessite trop de mémoire sur certaines séries (symbolisé par $-$). Les instances aléatoires binaires de la classe $\langle 40, 180, 84, 0.9\rangle$ ne comportent en moyenne que 12 3-cliques, ce qui explique pourquoi il est économique d'établir sCPC.

[1] `http://cpai.ucc.ie/06/Competition.html`

La table 2 fournit d'autres résultats représentatifs, instance par instance. Une nouvelle fois, il apparaît que, hormis l'instance $fapp$-01-200-4, sCDC-1 surclasse largement les algorithmes qui établissent sCPC. Il y a une différence par un ordre de magnitude sur l'instance $haystack$-40 et par presque deux ordres de magnitude sur l'instance $knights$-50-5. La performance relativement mauvaise de sCDC-1, en terme de temps cpu, sur l'instance $fapp$-01-200-4 peut une nouvelle fois être expliquée par la densité très faible (D est seulement à 0.5%) et le petit nombre de 3-cliques. Cependant, l'amélioration en terme de filtrage est tout à fait significative. De plus, sCDC-1 est bien moins consommatrice de mémoire que SAC et les algorithmes PC. La différence existante avec AC3^{rm} provient du fait qu'un plus grand nombre de relations peut être partagé entre contraintes lorsque les relations ne sont pas modifiées.

Pour finir, nous avons comparé la performance de l'algorithme générique état-de-l'art MAC avec et sans sCDC établi en pré-traitement sur des instances difficiles RLFAP (Radio Link Frequency Assignment Problem). Même si des techniques (redémarrages, enregistrement de nogoods, etc.) permettent de résoudre plus efficacement ces instances, nous ne les employons pas ici de manière à observer l'impact réel (sur la recherche) de sCDC établi en pré-traitement. La table 3 montre que pour les instances les plus difficiles, sCDC en pré-traitement est payant : MAC tout seul est environ 40% plus lent que sCDC+MAC, et visite presque 2 fois plus de noeuds.

	AC3^{rm}	SAC-SDS	sCPC8 / sCPC2001	sCDC-1
driverlogw-09 ($K = 233,834$; $D = 8\%$)				
cpu	1.60	48.42	33.84 / 36.52	10.83
mem	14	87	59 / 155	23
λ	369,736	147,115	306,573	18,958
haystack-40 ($K = 395,200$; $D = 2\%$)				
cpu	9.64	–	580.48 / –	55.91
mem	19	–	209 / –	107
λ	48,670,518	–	48,670,518	48,670,518
knights-50-5 ($K = 10$; $D = 100\%$)				
cpu	12.38	34.43	1759 / –	21.49
mem	5	163	29 / –	19
λ	31,331,580	0	0	0
pigeons-50 ($K = 19,600$; $D = 100\%$)				
cpu	1.38	2.85	33.82 / 44.52	2.7
mem	2	12	9 / 636	5
λ	2,881,200	2,881,200	2,881,200	2,881,200
qcp-25-264-0 ($K = 43,670$; $D = 5\%$)				
cpu	2.28	6.08	8.15 / 10.49	2.08
mem	8	210	29 / 215	21
λ	77,234	77,234	76,937	76,937
qwh-25-235-0 ($K = 35,700$; $D = 4.5\%$)				
cpu	1.87	5.62	7.09 / 9.05	2.56
mem	7	183	26 / 173	19
λ	56,721	56,721	56,380	56,380
fapp01-200-4 ($K = 247$; $D = 0.5\%$)				
cpu	10.73	–	16.05 / 18.63	104.05
mem	15	–	22 / 254	17
λ	3,612,163	–	3,317,135	2,117,575
scen-11 ($K = 13,775$; $D = 1.7\%$)				
cpu	2.87	–	85.82 / 78.49	9.78
mem	5	–	22 / 426	16
λ	5,434,107	–	4,829,442	4,828,650

TAB. 2 – Résultats expérimentaux (mem en MiB)

$Instance$		MAC	$sCDC$+MAC
scen11-f8	cpu	8.08	14.31
	$nodes$	14,068	4,946
scen11-f5	cpu	259	225
	$nodes$	$1,327K$	$680K$
scen11-f3	cpu	2,338	1,725
	$nodes$	$12M$	$5,863K$
scen11-f2	cpu	7,521	5,872
	$nodes$	$37M$	$21M$
scen11-f1	cpu	17,409	13,136
	$nodes$	$93M$	$55M$

TAB. 3 – Impact de sCDC en pré-traitement de MAC

6 Conclusion

Les solveurs proposés pour la satisfaction de contraintes sont généralement construits sur la base d'un algorithme de recherche systématique ou locale. Effectuer des inférences, autant que possible, lors d'une phase de pré-traitement peut grandement améliorer leurs performances. Par exemple, établir la singleton consistance d'arc sur des réseaux fortement structurés peut s'avérer payant. Malheureusement, les algorithmes qui établissent SAC ne sont pas capables de prendre en compte une grande part de l'information qui est disponible lorsqu'ils sont exécutés. En effet, si une valeur Y_b est éliminée après avoir effectué l'assignation $X \leftarrow a$ et établi AC, nous sommes certain que $X = a$ et $Y = b$ ne sont pas compatibles.

Dans cet article, nous avons proposé de prendre en compte des inférences de ce type de manière conservative, i.e. sans ajouter aucune nouvelle contrainte. Nous avons introduit une nouvelle consistance appelée consistance duale (DC) et nous sommes focalisés sur sa variante conservative CDC. Il a été montré en particulier que CDC est une consistance de relation qui est plus forte que PC conservative (CPC), et qu'établir la CDC forte (i.e. établir à la fois CDC et AC) peut être réalisé de manière tout à fait simple et naturelle. Les résultats expérimentaux que nous avons obtenus sur un vaste échantillon de problèmes montrent clairement l'intérêt pratique de CPC sur les réseaux de forte densité.

Remerciements

Ce papier a été soutenu par le programme Cocoa de la Région Nord/Pas-de-Calais, le CNRS et par l'IUT de Lens.

Références

[1] C. Bessiere. Constraint propagation. Technical report, LIRMM, Montpellier, 2006.

[2] C. Bessiere and R. Debruyne. Optimal and suboptimal singleton arc consistency algorithms. In *Proceedings of IJCAI'05*, pages 54–59, 2005.

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

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

[5] R. Debruyne. A strong local consistency for constraint satisfaction. In *Proceedings of ICTAI'99*, pages 202–209, 1999.

[6] R. Debruyne and C. Bessiere. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14 :205–230, 2001.

[7] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[8] R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173(1) :283–308, 1997.

[9] C. Lecoutre and F. Hemery. A study of residual supports in arc consistency. In *Proceedings of IJCAI'07*, pages 125–130, 2007.

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

[11] J.J. McGregor. Relational consistency algorithms and their application in finding subgraph and graph isomorphisms. *Information Sciences*, 19 :229–250, 1979.

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

[13] K. Stergiou and T. Walsh. Inverse consistencies for non-binary constraints. In *Proceedings of ECAI'06*, pages 153–157, 2006.