Cohérences basées sur les valeurs en échec

Christophe Lecoutre, Olivier Roussel

Cohérences basées sur les valeurs en échec

Christophe Lecoutre **Olivier Roussel**

Université Lille-Nord de France, Artois, F-62307 Lens
CRIL - CNRS UMR 8188, F-62307 Lens
F-62307 Lens
{lecoutre,roussel}@cril.fr

Résumé

Dans le domaine de la satisfaction de contraintes, les inférences se basent en général sur certaines propriétés des réseaux de contraintes, appelées cohérences. Celles-ci permettent l'identification d'instanciations incohérentes, ou nogoods. Deux familles principales de cohérences ont été étudiées jusqu'ici : celles qui permettent de raisonner à partir des variables telles que la (i, j)-cohérence et celles qui permettent de raisonner à partir des contraintes telles que la (i, j)-cohérence relationnelle. Dans cet article, nous introduisons une nouvelle famille de cohérences basées sur le concept de valeur en échec (valeur éliminée au cours de la recherche). Cette famille est orthogonale aux précédentes.

Abstract

In constraint satisfaction, basic inferences rely on some properties of constraint networks, called consistencies, that allow the identification of inconsistent instantiations or nogoods. Two main families of consistencies have been introduced so far : those that permit to reason from variables such as (i, j)-consistency and those that permit to reason from constraints such as relational (i, j)-consistency. This paper introduces a new family of consistencies based on the concept of failed value (a value pruned during search). This family is orthogonal to previous ones.

1 Introduction

L'utilisateur d'un solveur de contraintes souhaite idéalement que le système soit suffisamment robuste et intelligent pour identifier et exploiter automatiquement toutes les propriétés d'une instance à résoudre. Typiquement, ces propriétés dépendent de la structure de l'instance et permettent de la résoudre plus facilement. Diverses approches sont explorées pour atteindre ce but : l'utilisation de cohérences fortes, les heuristiques adaptatives, l'apprentissage de nogoods et l'identification de symétries. Ces techniques permettent d'explorer efficacement l'espace de recherche en glanant de précieuses informations qui permettent d'élaguer l'arbre de recherche.

Généralement, on utilise une recherche arborescente pour résoudre les problèmes de satisfaction de contraintes (CSP). La recherche arborescente combine une exploration en profondeur d'abord pour instancier les variables avec un mécanisme de retour arrière pour gérer les impasses. Pendant la recherche, certaines valeurs sont identifiées comme incohérentes, i.e. elles ne peuvent figurer dans aucune solution. Nous appelons ces valeurs des valeurs en échec (FV pour Failed Values). On sait [14] que les valeurs en échec apportent une information : étant donnée une instance CSP binaire satisfiable P, pour tout couple (x, a) où x est une variable de P et a une valeur du domaine de x, s'il n'y a pas de solution assignant a à x, alors toute solution assigne nécessairement à au moins une autre variable y une valeur qui est incompatible avec (x, a). Cette propriété est utilisée pour décomposer une instance CSP de manière dynamique et itérée dans [14, 3].

Dans cet article, nous proposons d'utiliser les valeurs en échec de manière différente. En effet, certaines inférences sont possibles en raisonnant localement sur ces valeurs à un coût qui reste intéressant. Plus précisément, à partir des valeurs en échec, nous construisons une nouvelle famille de cohérences locales réduisant les domaines et montrons que cette famille est orthogonale (i.e. incomparable) avec les cohérences habituelles. Elles contribuent à la réduction de l'espace de recherche et permettent accessoirement une détection paresseuse d'une forme généralisée de la relation de substituabilité. Les cohérences basées sur les valeurs en échec s'intègrent aisément à n'importe quel moteur de propagation de contraintes et renforcent le pouvoir de filtrage de l'algorithme de recherche. Ces nouvelles cohérences peuvent contribuer à la mise au point de solveurs

plus robustes.

Après un rappel des définitions, cet article présente la *FV-cohérence d'arc* (AFVC) et un algorithme permettant de maintenir AFVC. Nous comparons ensuite AFVC à la substituabilité ainsi qu'aux cohérences usuelles. Enfin, nous montrons qu'une nouvelle famille de cohérences peut-être construite très naturellement à partir des valeurs en échec.

2 Définitions et notations

Un réseau de contraintes (CN pour *Constraint Network*) P se compose d'un ensemble fini de n variables, noté $vars(P)$, et d'un ensemble fini de e contraintes, noté $cons(P)$. Chaque variable x a un domaine associé noté $dom(x)$ qui contient l'ensemble fini de valeurs qui peuvent être assignées à x. Chaque contrainte c porte sur un ensemble de variables appelé *portée* de c et noté $scp(c)$. Une contrainte est définie par une relation notée $rel(c)$ qui contient l'ensemble des tuples autorisés pour les variables de $scp(c)$. L'*arité* d'une contrainte est le nombre de variables dans sa portée. Une contrainte *binaire* porte sur 2 variables tandis qu'une contrainte *n-aire* porte sur strictement plus de 2 variables. Pour une contrainte binaire c_{xy} telle que $scp(c_{xy}) = \{x, y\}$, on dit que (x, a) et (y, b) sont compatibles ssi $(a, b) \in rel(c_{xy})$. Lorsqu'il n'y a pas de contrainte entre x et y, toute valeur de x est dite compatible avec chaque valeur de y.

On considère donné un réseau initial de contraintes P^{init} et un réseau courant P dérivé de P^{init} en réduisant les domaines des variables. Le domaine initial d'une variable x est noté $dom^{init}(x)$ et le domaine courant est noté $dom^P(x)$ ou simplement $dom(x)$. Pour toute variable x, on a $dom(x) \subseteq dom^{init}(x)$ et on note $P \preceq P^{init}$. Une valeur courante de P est un couple (x, a) avec $x \in vars(P)$ et $a \in dom(x)$. Sans perte de généralité, les réseaux de contraintes considérés dans cet article ne contiendront ni contraintes d'arité 1, ni deux contraintes de même portée (il suffit de normaliser les réseaux [1, 4]).

Une instanciation I d'un ensemble $X = \{x_1, \ldots, x_k\}$ de variables est un ensemble $\{(x_1, a_1), \ldots, (x_k, a_k)\}$ tel que $\forall i, a_i \in dom^{init}(x_i)$; l'ensemble X des variables de I est noté $vars(I)$ et chaque valeur a_i est notée $I[x_i]$. Une instanciation I sur un réseau P est une instanciation d'un ensemble $X \subseteq vars(P)$; elle est complète lorsque $vars(I) = vars(P)$, et partielle dans le cas contraire. I est valide sur P ssi $\forall (x, a) \in I, a \in dom(x)$ $(= dom^P(x))$. Une instanciation I recouvre une contrainte c ssi $scp(c) \subseteq vars(I)$, et elle satisfait une contrainte c avec $scp(c) = (x_1, \ldots, x_r)$ ssi a) I recouvre c et b) le tuple $(a_1, \ldots, a_r)$, tel que $\forall i, a_i = I[x_i]$, est autorisé par c. Une instanciation I sur un réseau P est localement cohérente ssi a) I est valide sur P et b) chaque contrainte de P recouverte par I est satisfaite par I. Si ce n'est pas le cas, I est localement incohérente. Une solution de P est une instanciation complète de P qui est localement cohérente. Une instanciation I sur un réseau P est globalement incohérente (ou également appelé *nogood*) ssi elle ne peut être étendue pour devenir une solution de P. Elle est globalement cohérente dans le cas contraire.

Une solution d'un CN est une assignation d'une valeur à chaque variable de sorte que chaque contrainte soit satisfaite. Un CN est *satisfiable* ssi il possède au moins une solution. Le problème de satisfaction de contraintes (CSP) est le problème NP-dur de déterminer si un réseau donné est satisfiable ou pas. Une instance CSP est déterminée par un CN qui est résolu soit en identifiant une solution, soit en prouvant l'insatisfiabilité. Pour résoudre une instance CSP, une recherche en profondeur d'abord avec retour arrière est souvent utilisée. À chaque étape de la recherche, une variable se voit assigner une valeur. S'ensuit alors un processus de filtrage nommé propagation de contraintes qui permet d'éviter des assignations futures qui seraient incohérentes avec les choix précédents. Typiquement, les algorithmes de propagation de contraintes utilisent des propriétés des réseaux qui permettent d'éliminer des valeurs qui ne peuvent plus apparaître dans une solution. Ces propriétés sont appelées cohérences réduisant les domaines (domain-filtering consistencies) [8, 5].

Nous rappelons brièvement les principales cohérences usuelles. Une instanciation I est un support pour une valeur (x, a) sur une contrainte c portant sur x ssi I est valide, $I[x] = a$ et I satisfait c. Une valeur (x, a) de P est GAC-cohérente (GAC pour Generalized Arc Consistency) ssi il existe un support pour (x, a) sur toute contrainte de P portant sur x. P est GAC-cohérente ssi toute valeur de P est GAC-cohérente. Pour les réseaux binaires, la propriété GAC est nommée AC (cohérence d'arc). Pour tout réseau P, il existe un plus grand sous-réseau de P qui est GAC-cohérent. Ce sous-réseau est la clôture GAC de P que l'on notera $GAC(P)$: cette clôture est équivalente à P et telle que $GAC(P) \preceq P$. Si le domaine d'une variable de P devient vide, P est trivialement insatisfiable (noté $P = \bot$). $P|_{x=a}$ est le réseau obtenu à partir du réseau P en retirant du domaine de x toute valeur $b \neq a$. Une valeur (x, a) de P est SAC-cohérente (SAC pour Singleton Arc Consistency) ssi $GAC(P|_{x=a}) \neq \bot$. P est SAC-cohérente ssi toutes ses valeurs sont SAC-cohérentes.

Une cohérence réduisant les domaines permet d'identifier et d'éliminer des valeurs incohérentes. Un pré-ordre [8] peut être défini sur ces cohérences comme suit. Soient ϕ et ψ deux cohérences. ϕ est plus forte que ψ, noté $\phi \trianglerighteq \psi$, ssi chaque fois que ϕ est vérifiée sur un réseau P, ψ est aussi vérifiée sur P. ϕ est strictement plus forte que ψ, noté $\phi \triangleright \psi$ ssi $\phi \trianglerighteq \psi$ et il existe un réseau P tel que ψ est vérifiée sur P tandis que ϕ ne l'est pas. Quand des cohérences ne peuvent être classées (aucune n'est plus forte que l'autre), elles sont dites *incomparables*. Pour les cohé-

rences usuelles sur les réseaux binaires, on sait que : SAC $\rhd$ MaxRPC $\rhd$ PIC $\rhd$ AC.

3 Cohérences élémentaires basées sur les valeurs en échec

Dans cette section, nous définissons deux nouvelles cohérences élémentaires. La première identifie des nogoods de taille quelconque tandis que la seconde identifie des valeurs incohérentes (nogoods de taille 1). Ces deux cohérences sont basées sur la notion de valeur en échec (FV pour Failed Value). Les valeurs en échec apportent de l'information.

Lemme 1 (dérivé directement de [14])**.** *Si une valeur (x, a) d'un réseau P est globalement incohérente alors chaque solution S de P est telle que $S[x/a]$ viole au moins une contrainte de P impliquant x.*

Démonstration. $S[x/a]$ n'est pas une solution de P puisque (x, a) est globalement incohérente. Cela signifie que au moins une contrainte de P n'est pas satisfaite par $S[x/a]$. Mais nous savons que chaque contrainte c de P qui n'implique pas x est satisfaite par $S[x/a]$ car la restriction de $S[x/a]$ sur $scp(c)$ est exactement la restriction de S sur $scp(c)$. En conséquence, au moins une contrainte de P impliquant x n'est pas satisfaite par $S[x/a]$. □

Définition 1. *Une valeur en échec de P est une valeur d'un réseau $P' \succ P$ qui est globalement incohérente pour P' et qui n'apparaît plus dans P.*

En pratique, une valeur en échec est une valeur retirée d'un réseau parce qu'elle a été prouvée globalement incohérente. Une valeur en échec peut être identifiée en utilisant des techniques d'inférence et/ou de recherche.

Définition 2. *Soit (x, a) une valeur de P. Pour toute contrainte c de P portant sur x, l'ensemble conflit de (x, a) pour c, noté $\chi(c, x, a)$, est l'ensemble des instanciations valides I sur P de $scp(c) \setminus \{x\}$ telles que $I \cup \{(x, a)\}$ ne satisfait pas c. L'ensemble conflit de (x, a) pour P est $\chi(x, a) = \cup_{c \in cons(P) | x \in scp(c)} \chi(c, x, a)$.*

Quel que soit l'ensemble conflit χ, $vars(\chi) = \cup_{I \in \chi} vars(I)$. La figure 1 représente un réseau avec une contrainte binaire entre w et x et une contrainte ternaire entre w, y et z. Dans chacune des figures de cet article, des lignes pleines (respectivement en pointillés) représentent des tuples autorisés (respectivement interdits). L'absence d'arêtes entre deux variables x et y signifie qu'il n'y a aucune contrainte binaire portant sur x et y. Dans cet exemple, $\chi(w, a) = \{\{(x, b)\}, \{(y, a), (z, a)\}\}$ et $\chi(w, c) = \{\{(x, b)\}, \{(x, c)\}, \{(y, c), (z, c)\}\}$.

La définition suivante relie les instanciations aux valeurs en échec.

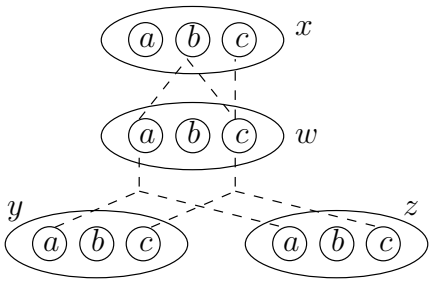

FIG. 1 – Exemple d'ensembles conflit.

Définition 3. *Soit (w, p) une valeur en échec de P et I une instanciation sur P. I recouvre (w, p) ssi $vars(\chi(w, p)) \subseteq vars(I)$. I satisfait (w, p) ssi $\exists J \in \chi(w, p) \mid J \subseteq I$.*

Une valeur en échec satisfaite par une instanciation n'est pas nécessairement recouverte par elle. Cependant, quand une valeur en échec est recouverte par une instanciation et que cette valeur n'est pas satisfaite, un nogood peut être identifié. FVC est utilisé pour Failed Value Consistency.

Définition 4 (FVC)**.** *Une instanciation I valide sur P est FVC-cohérente vis à vis d'une valeur en échec (w, p) de P ssi soit (w, p) n'est pas recouverte par I soit (w, p) est satisfaite par I. I est FVC-cohérente ssi elle est FVC-cohérente vis à vis de toute valeur en échec de P ; elle est dite FVC-incohérente sinon.*

Imaginons que (w, c) de la figure 1 soit une valeur en échec. $I = \{(x, a), (y, c), (z, c)\}$ est une instanciation qui satisfait (w, c) car $\{(y, c), (z, c)\} \in \chi(w, c) \subset I$. $I' = \{(x, a)\}$ ne satisfait pas (w, c) mais est malgré tout FVC-cohérente car (w, c) n'est pas recouverte par I'. $I'' = \{(x, a), (y, a), (z, a)\}$ est FVC-incohérente car (w, c) est à la fois recouverte par I'' et non satisfaite par I''.

Proposition 1. *Toute instanciation FVC-incohérente est globalement incohérente.*

Démonstration. Par définition, quand (w, p) est une valeur en échec de P, il n'y a aucune solution de P contenant (w, p). Pour toute solution S de P, soit $S[w/p]$ l'instanciation complète obtenue à partir de S en y remplaçant la valeur donnée à w par la valeur p. $S[w/p]$ ne peut être une solution car au moins une contrainte portant sur w est nécessairement falsifiée (voir le lemme 1). Quand une instanciation valide I est FVC-incohérente vis à vis de (w, p), c'est qu'il n'y a aucun moyen d'étendre I en une instanciation complète I' telle que $I'[w/p]$ viole au moins une contrainte portant sur w. De ce fait, I est un nogood. □

En d'autres termes, des nogoods peuvent être déduits d'autres nogoods (les valeurs en échec). Ces nogoods déduits peuvent être de taille quelconque comme le montre la figure 2. Dans cet exemple, il y a une valeur en échec

(w, p) et trois contraintes binaires portant sur w. Une instanciation valide de $\{x, y, z\}$ est globalement incohérente si elle contient uniquement des valeurs compatibles avec (w, p). Autrement dit, chaque tuple de $C_x \times C_y \times C_z$ correspond à un nogood (de taille 3).

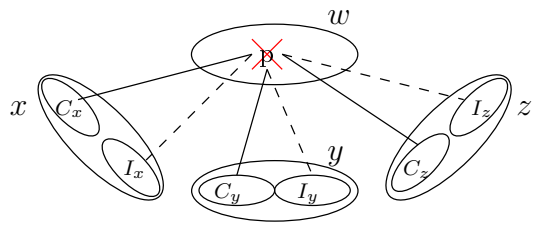

FIG. 2 – Une valeur en échec (w, p), les valeurs compatibles avec (w, p) sont dans C_x, C_y et C_z et les valeurs incompatibles sont dans I_x, I_y et I_z.

Pour les réseaux binaires, une solution directe[1] pour utiliser ces nogoods déduits est de poster, pour chaque valeur en échec (w, p), une contrainte n-aire portant sur $vars(\chi(w, p))$ et qui interdit toute instanciation FVC-incohérente vis à vis de (w, p). Par exemple, dans le cas de la figure 2, on obtient une contrainte ternaire c_{xyz} telle que $rel(c_{xyz}) = dom(x) \times dom(y) \times dom(z) \setminus C_x \times C_y \times C_z$. Il est possible d'utiliser des algorithmes de filtrage efficaces (propagateurs) pour maintenir GAC sur ce type de contraintes. Cependant, cette solution a différents inconvénients : il faut modifier le réseau de contraintes dynamiquement, la généralisation au cas n-aire est complexe et le filtrage est limité aux variables de la contrainte postée.

Cette dernière remarque (filtrage limité) est moins vraie si l'on utilise une cohérence plus forte que GAC telle que SAC. Cependant, les solveurs actuels n'utilisent guère SAC durant la recherche. Intuitivement, on peut utiliser les valeurs en échec pour filtrer davantage en raisonnant sur chaque valeur et les ensembles conflit de chaque valeur en échec. En particulier, on peut définir à partir des valeurs en échec une cohérence réduisant les domaines comme suit. AFVC est utilisé pour Arc Failed Value Consistency.

Définition 5 (AFVC)**.**
- *Une valeur (x, a) de P est AFVC-cohérente vis à vis d'une valeur en échec (w, p) de P ssi (x, a) peut être étendue en une instanciation localement cohérente satisfaisant (w, p).*
- *Une valeur (x, a) de P est AFVC-cohérente ssi (x, a) est AFVC-cohérente vis à vis de toute valeur en échec de P ; elle est dite AFVC-incohérente sinon.*
- *P est AFVC-cohérente ssi chaque valeur de P est AFVC-cohérente.*

Dans le premier point de la définition, il faut noter qu'on peut avoir $x = w$. Dans ce cas, on a nécessairement $a \neq p$ puisque (x, a) est une valeur courante tandis que (w, p) a été éliminée. Notons que AFVC peut être perçue comme une cohérence locale puisqu'il suffit de raisonner à partir de l'ensemble conflit de chaque valeur en échec. En particulier pour les réseaux binaires, une valeur (x, a) est AFVC-cohérente vis à vis d'une valeur en échec (w, p) ssi (x, a) est compatible avec une valeur valide dans $\chi(w, p)$. Un algorithme pour établir AFVC et basé sur cette simple observation est présenté plus loin.

Le résultat suivant est important :

Proposition 2. *Toute valeur AFVC-incohérente est globalement incohérente.*

Démonstration. Soit (x, a) une valeur courante de P qui est AFVC-incohérente vis à vis d'une valeur en échec (w, p) de P. Supposons qu'il existe une solution S de P telle que $S[x] = a$. Nécessairement, puisque (x, a) est AFVC-incohérente vis à vis de (w, p), $S[w/p]$ ne viole aucune contrainte portant sur w, et donc est une solution de P. Cela contredit le fait que (w, p) soit globalement incohérente (puisque c'est une valeur en échec), et donc notre hypothèse. On en déduit que (x, a) est globalement incohérente. □

La figure 3 présente un fragment d'un réseau de contraintes qui peut être complété pour faire en sorte que les cohérences usuelles (cohérence d'arc,...) soient vérifiées. On suppose que (w, p) est une valeur en échec et que $\chi(w, p) = \{\{(x, a)\}, \{(y, c)\}\}$. Il est facile de vérifier que (w, a) et (z, a) sont AFVC-incohérentes. En effet, (w, a) et (z, a) ne sont compatibles avec aucune valeur de $\chi(w, p)$.

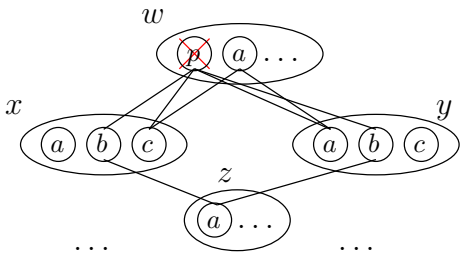

FIG. 3 – Illustration de AFVC

Quand on se restreint à des nogoods de taille 1, FVC est plus faible que AFVC (preuve omise). On a :

Proposition 3. *Soit (x, a) une valeur de P. Si $I = \{(x, a)\}$ est FVC-incohérente alors (x, a) est AFVC-incohérente.*

Enfin, on peut montrer que AFVC vérifie les propriétés (cf. [1, 4]) qui permettent de définir la clôture AFVC.

Proposition 4. *Pour tout réseau P, il existe un plus grand réseau AFVC-cohérent équivalent à P, appelé la clôture AFVC de P et noté $AFVC(P)$, tel que $AFVC(P) \preceq P$.*

[1] Une approche similaire bien que différente du point de vue opérationnel est de décomposer le problème [14]

$AFVC(P)$ peut s'obtenir en éliminant itérativement, dans un ordre quelconque, les valeurs qui ne sont pas AFVC-cohérentes.

4 Algorithme pour établir AFVC sur les réseaux binaires

À partir d'un réseau binaire P et d'un ensemble D de valeurs en échec, la procédure établirAFVC (algorithme 1) calcule $AFVC(P)$ ou lance une exception si un domaine devient vide. Les structures de données utilisées sont les suivantes. Pour chaque valeur en échec (w,p), $\chi(w,p)$ est un tableau de valeurs indexées de 1 à $length(\chi(w,p))$. Ces valeurs correspondent aux instanciations (de taille 1 puisque P est binaire) de l'ensemble conflit de (w,p). À chaque couple composé d'une valeur (x,a) de P et d'une valeur en échec (w,p) de D, le tableau à deux dimensions $last$ associe un entier qui est l'indice de la dernière valeur de $\chi(w,p)$ identifiée comme compatible avec (x,a) : $last[(x,a)][(w,p)]$ donne la position du dernier support AFVC trouvé pour (x,a) sur (w,p). Pour chaque valeur (y,b) de P, $S(y,b)$ est une liste stockant les couples (valeur,valeur en échec) pour lesquels (y,b) est le dernier support AFVC identifié. Les structures S et $last$ sont inspirées de celles de AC6 et AC2001 (voir [4]).

La procédure initialiser initialise les structures de données : les ensembles conflit $\chi(w,p)$ sont construits et les listes $S(x,a)$) sont vidées. La procédure établirAFVC tente d'identifier un support AFVC pour chaque couple (valeur,valeur en échec). Si aucun support n'est trouvé pour (x,a) sur une valeur en échec, a est retiré de $dom(x)$ et ajouté à la file de propagation Q. Chaque valeur $(y,b) \in Q$ est propagée : un nouveau support est recherché pour chaque couple de $S(y,b)$ (à partir de la dernière position enregistrée).

```
1  procedure initialiser(P : CN, D : ensemble de valeurs en échec)
2  begin
3      foreach valeur en échec (w,p) ∈ D do
4          χ(w,p) ← ∅
5          foreach voisin x de w do
6              foreach valeur a ∈ dom(x) do
7                  if (w,p) et (x,a) sont incompatibles then
8                      ajouter (x,a) à χ(w,p)
9          if χ(w,p) = ∅ then
10             throw INCOHÉRENCE
11     foreach valeur (x,a) de P do
12         S(x,a) ← ∅
13 end
```

```
1  procedure rechercheSupportFV((x,a) : valeur, (w,p) : valeur en échec)
2  begin
3      position ← last[(x,a)][(w,p)] + 1
4      while position ≤ length(χ(w,p)) do
5          (y,b) ← χ(w,p)[position]
6          if b ∈ dom(y) ∧ (x,a) et (y,b) sont compatibles then
7              last[(x,a)][(w,p)] ← position
8              ajouter ((x,a),(w,p)) à S(y,b)
9              return
10         position ← position + 1
11     retirer a de dom(x)
12     if dom(x) = ∅ then
13         throw INCOHÉRENCE
14     ajouter (x,a) à Q
15 end
16 procedure établirAFVC(P : CN, D : ensemble de valeurs en échec)
17 begin
18     Q ← ∅
19     foreach valeur (x,a) de P do
20         foreach valeur en échec (w,p) ∈ D do
21             last[(x,a)][(w,p)] ← 0
22             rechercheSupportFV((x,a),(w,p))
23     while Q ≠ ∅ do
24         extraire (y,b) de Q
25         foreach ((x,a),(w,p)) de S(y,b) do
26             rechercheSupportFV((x,a),(w,p))
27         S(y,b) ← ∅
28 end
```

Algorithm 1: Etablir AFVC

Dans le pire des cas, l'algorithme a une complexité en espace de $O(pM + pnd + pnd) = O(pnd)$ et une complexité en temps de $O(pMnd)$, avec n le nombre de variables, d la taille du plus grand domaine, p le nombre de valeurs en échec ($p = |D|$) et M la taille maximale d'un ensemble conflit ($M = max_{(w,p)\in D}|\chi(w,p)|$). Dans le détail, initialiser a une complexité en temps de $O(pnd)$, la complexité cumulée de rechercheSupportFV pour chaque couple (valeur,valeur en échec) est $O(M)$, et il y a $O(pnd)$ couple différent. Par définition, $p < nd$ et $M < nd$ donc $pMnd < n^3d^3$. En pratique, il est sans doute intéressant de borner par une constante le nombre de valeurs en échec et/ou la taille maximale des ensembles conflit pour se concentrer sur les valeurs en échec prometteuses. En bornant ces deux paramètres, la complexité devient $O(nd)$.

Ces complexités semblent satisfaisantes (par exemple en comparaison de $O(ed^2)$ pour un algorithme AC optimal en temps). Il est facile d'adapter cet algorithme pour l'utiliser

à chaque étape d'une recherche arborescente (la complexité reste la même sur une branche de l'arbre).

5 Substiuabilité et cohérences usuelles

La substituabilité de voisinage est une forme affaiblie de substituabilité [12] qui peut être rapprochée des cohérences basées sur les valeurs en échec. Une valeur $a \in dom(x)$ est voisin-substituable à une valeur $b \in dom(x)$ ssi pour chaque contrainte c portant sur x et tout support I pour (x, b) sur c, $I[x/a]$ est un support pour (x, a) sur c. La satisfiabilité des instances est préservée quand les valeurs voisin-substituables à une autre sont retirées. La sustituabilité de voisinage est utilisable pour réduire l'espace de recherche pour une instance tout en préservant la satisfiabilité [2]. Par exemple, elle peut être exploitée en tant qu'opérateur de réduction en appliquant une séquence convergente de substitutions de voisinage [7]. Cette notion est clairement liée au concept de symétrie [6]. La proposition suivante établit un lien avec AFVC.

Proposition 5. *Si une valeur (x, a) est voisin-substituable à une valeur (x, b) dans un réseau $P' \succ P$ et si (x, a) est une valeur en échec de P et (x, b) une valeur courante de P, alors (x, b) est AFVC-incohérente.*

Démonstration. La définition de substituabilité de voisinage peut être reformulée en : (x, a) est voisin-substituable à (x, b) ssi $\chi(x, a) \subseteq \chi(x, b)$. Si (x, a) est une valeur en échec de P, alors il n'est pas possible d'étendre (x, b) en une instanciation cohérente qui satisfasse (x, a). (x, b) est donc AFVC-incohérente. □

On peut voir AFVC comme un mécanisme paresseux et dynamique permettant d'identifier les valeurs substituables (et globalement incohérentes). Mais en fait, AFVC permet aussi d'identifier des valeurs incohérentes qui ne sont pas substituables. En fait, une valeur (x, b) est AFVC-incohérente si l'ensemble conflit de (x, b) est inclus dans l'ensemble conflit d'une quelconque valeur en échec. À la différence de la substituabilité de voisinage qui ne considère que les valeurs d'une même variable, AFVC peut identifier des valeurs dans des variables différentes et est donc plus générale de ce point de vue. Un exemple est donné en figure 3 : (w, p) est substituable à (w, a) mais pas à (z, a) puisque $w \neq z$.

Proposition 6. *AFVC et AC sont incomparables.*

Démonstration. Trivialement, AFVC $\not\vdash$ AC car il suffit de considérer un réseau qui n'est pas arc-cohérent et qui ne contient aucune valeur en échec. Par ailleurs, AC $\not\vdash$ AFVC. En effet, la figure 4 présente un réseau qui est arc-cohérent mais pas AFVC-cohérent. Plus précisément, le réseau P situé en haut de la figure est clairement arc-cohérent et AFVC-cohérent (puisqu'il n'y a pas de valeurs en échec). Supposons maintenant que la valeur (x, c) a été identifiée comme globalement incohérente (durant la recherche par exemple) : (x, c) est alors éliminé de $dom(x)$ pour aboutir au réseau P' situé en bas de la figure. Par définition, (x, c) est une valeur en échec, et on a $\chi(x, c) = \{\{(y, b)\}, \{(z, b)\}\}$. On peut observer que même si P' est arc-cohérent, P' n'est pas AFVC-cohérent car (x, a) n'est pas AFVC-cohérent. □

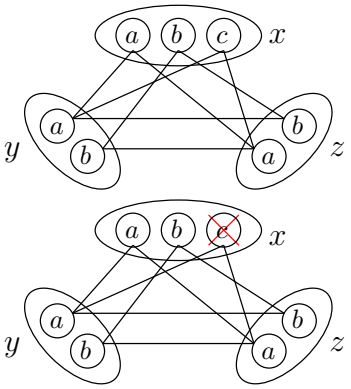

FIG. 4 – AC $\not\vdash$ AFVC

Il est intéressant de constater que AFVC est incomparable avec la plupart des cohérences réduisant les domaines. Plus précisément, AFVC est incomparable avec les cohérences «usuelles», i.e. les cohérences locales ϕ qui vérifient trois propriétés de base : a) ϕ est vérifiée sur tout réseau ne contenant que des contraintes universelles (une contrainte est universelle si elle est satisfaite par toute instanciation valide des variables dans sa portée), b) ϕ est vérifiée sur un réseau ssi elle est vérifiée sur chacune des composantes connexes du réseau, et c) il existe des réseaux insatisfiables pour lesquels ϕ est vérifiée. Par exemple, (G)AC, SAC, PIC, ... sont usuelles, mais la cohérence globale (définie comme toute instanciation localement cohérente peut être étendue en une solution) ne l'est pas.

Proposition 7. *AFVC est incomparable avec les cohérences usuelles.*

Démonstration. Soit ϕ une cohérence locale usuelle, un réseau P_1 qui ne contient que des contraintes universelles (donc satisfiable) et un réseau P_2 défini sur un jeu différent de variables qui soit insatisfiable mais néanmoins ϕ-cohérent. On considère le problème $P = P_1 \cup P_2$. Puisque P_2 est insatisfiable, toute valeur (x, a) de P_1 sera identifiée comme une valeur en échec lors de la recherche. Soit (x, a) l'une des valeurs en échec de P_1. Comme P_1 ne contient que des contraintes universelles, $\chi(x, a) = \varnothing$ et donc, aucune instanciation ne peut satisfaire la valeur en échec (x, a). De ce fait, P n'est pas AFVC-cohérent. En revanche, ϕ est vérifié sur P (conséquence des hypothèses), et donc $\phi \not\vdash$ AFVC. De l'autre côté, AFVC $\not\vdash \phi$: il suffit

de choisir un réseau P initial (donc sans valeurs en échec) et tel que ϕ ne soit pas vérifié. Par conséquent, AFVC et ϕ sont incomparables. □

6 Une hiérarchie de cohérences basées sur les valeurs en échec

Dans [11], Freuder a introduit la famille des (i, j)-cohérences. De manière informelle, un réseau de contraintes est (i, j)-cohérent ssi toute instanciation localement cohérente d'un ensemble de i variables peut être étendue en une instanciation localement cohérente contenant n'importe quel ensemble de j variables additionnelles. La cohérence d'arc, la cohérence de chemin [19, 18] et la cohérence de chemin inverse (PIC) [13] appartiennent toutes à cette famille puisqu'elles correspondent respectivement à la $(1, 1)$-cohérence, la $(2, 1)$-cohérence et la $(1, 2)$-cohérence. Une autre famille importante de cohérences basées cette fois-ci sur les contraintes est celle des (i, m)-cohérences relationnelles [9]. Informellement, un réseau est (i, m)-cohérent relationnel ssi pour chaque ensemble C de m contraintes avec $Y = \cup_{c \in C} scp(c)$ et tout ensemble $X \subseteq Y$ de i variables, toute instanciation localement cohérente de X peut être étendue en une instanciation valide de Y satisfaisant chaque contrainte de C. La cohérence d'arc généralisée et la cohérence de chemin inverse relationnelle [5] correspondent respectivement à la $(1, 1)$-cohérence relationnelle et la $(1, 2)$-cohérence relationnelle.

En s'inspirant de ces schémas, nous proposons une nouvelle famille de cohérences fondées sur la notion de valeurs en échec.

Définition 6 (FV-(i, p)-cohérence)**.** *P est FV-(i, p)-cohérent ssi pour chaque ensemble X de i variables de P et tout ensemble Y de p valeurs en échec de P, toute instanciation localement cohérente de X peut être étendue en une instanciation localement cohérente satisfaisant chaque valeur en échec de Y.*

De nombreuses cohérences découlent de cette définition générale : la FV-cohérence d'arc (AFVC) est la FV-$(1, 1)$-cohérence ; la FV-cohérence de chemin (PFVC) est la FV-$(2, 1)$-cohérence et la FV cohérence de chemin inverse (PIFVC) est la FV-$(1, 2)$-cohérence.

Par analogie avec les cohérences MaxRPC et SAC (voir par exemple [4]) basées sur les variables, nous définissons deux nouvelles cohérences.

Définition 7 (MaxFVC)**.** *Une valeur (x, a) de P est MaxFVC-cohérente ssi pour chaque valeur en échec (w, p) de P, (x, a) peut être étendue en une instanciation localement cohérente I satisfaisant (w, p) et telle que pour toute valeur en échec additionnelle (w', p') de P, I peut être étendue en une instanciation localement cohérente satisfaisant (w', p').*

Définition 8 (SAFVC)**.** *Une valeur (x, a) de P est SAFVC-cohérente ssi $AFVC(P|_{x=a}) \neq \bot$.*

Proposition 8. *PIFVC $\rhd$ AFVC et SAFVC $\rhd$ AFVC.*

Démonstration. Il découle des définitions que PIFVC $\trianglerighteq$ AFVC et SAFVC $\trianglerighteq$ AFVC. La figure 5 présente un exemple montrant l'ordre strict. Le réseau P a deux valeurs en échec (w, p), et (x, p). De plus, $\chi(w, p) = \{\{(y, b)\}, \{(z, b)\}\}$ et $\chi(x, p) = \{\{(y, a)\}, \{(z, b)\}\}$. P est AFVC-cohérent mais n'est ni SAFVC-cohérent, ni PIFVC-cohérent. En effet, (z, a) ne peut être étendu à une instanciation satisfaisant les deux valeurs en échec simultanément et $AFVC(P|_{z=a}) = \bot$. □

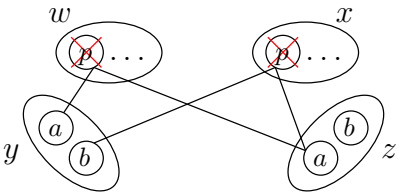

FIG. 5 – AFVC est vérifiée mais ni SAFVC, ni PIFVC ne le sont.

Proposition 9. *MaxFVC $\rhd$ PIFVC.*

Démonstration. Par définition, MaxFVC $\trianglerighteq$ PIFVC. La figure 6 permet de montrer l'ordre strict. Le réseau P possède trois valeurs en échec (w, p), (x, p) et (y, p). P est PIFVC-cohérent. En effet, chaque valeur de u (resp. v) est compatible avec (v, c) (resp. (u, c)) qui satisfait les trois valeurs en échec. Pour z, (z, b) est clairement PIFVC-cohérent tandis que pour (z, a), les valeurs en échec (w, p) et (x, p) sont satisfaites par $\{(z, a), (u, a), (v, a)\}$, les valeurs en échec (w, p) et (y, p) par $\{(z, a), (u, a), (v, b)\}$ et les valeurs en échec (x, p) et (y, p) par $\{(z, a), (u, b), (v, b)\}$. P n'est pas MaxFVC-cohérent. En effet, $\{(z, a), (u, b)\}$ et $\{(z, a), (v, a)\}$ sont les seules extensions de (z, a) qui satisfont (x, p). Cependant, pour $\{(z, a), (u, b)\}$, il est impossible de trouver une extension qui satisfasse (w, p) et pour $\{(z, a), (v, a)\}$, il est impossible de trouver une extension qui satisfasse (y, p). □

On peut aussi montrer que PIFVC et SAFVC d'une part, et MaxFVC et SAFVC d'autre part, sont incomparables.

Proposition 10. *PIFVC et SAFVC sont incomparables.*

Démonstration. Le réseau P de la figure 7 est SAFVC-cohérent mais pas PIFVC-cohérent. La valeur (z, a) ne peut être étendue pour satisfaire les deux valeurs en échec (x, p) et $y, p)$. Donc, le réseau est PIFVC-incohérent. De ce fait, SAFVC n'est pas plus fort que PIFVC. Par ailleurs,

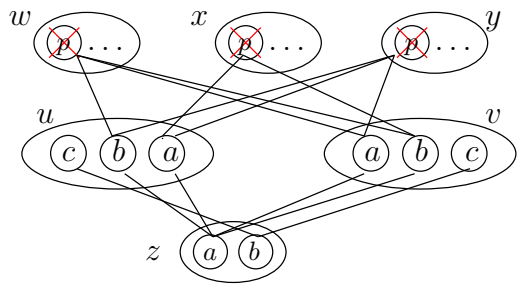

FIG. 6 – SAFVC et PIFVC sont vérifiées mais MaxFVC ne l'est pas.

la figure 6 représente un réseau qui n'est pas MaxFVC-cohérent mais qui est SAFVC-cohérent. Puisque MaxFVC est plus fort que PIFVC, on en déduit que PIFVC n'est pas plus fort que SAFVC. Donc, PIFVC et SAFVC sont incomparables. □

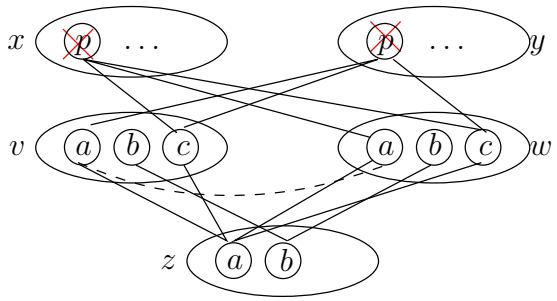

FIG. 7 – SAFVC est vérifié mais PIFVC ne l'est pas. Le seul tuple interdit entre v et w est (a, a).

Proposition 11. *MaxFVC et SAFVC sont incomparables.*

Démonstration. Le réseau de la figure 6 n'est pas MaxFVC-cohérent mais est SAFVC-cohérent. De ce fait, SAFVC n'est pas plus fort que MaxFVC. Considérons maintenant le réseau de la figure 7 auquel on apporte deux modifications : la valeur (z, b) est retirée du domaine initial de la variable et l'incompatibilité entre (v, a) et (w, a) est également effacée. Le réseau résultant est MaxFVC-cohérent mais pas SAFVC-cohérent. En effet, $AFVC(P|_{w=c}) = \bot$. □

La figure 8 résume les relations entre les différentes cohérences basées sur les valeurs en échec.

7 Conclusion

Cet article montre comment les valeurs détectées comme globalement incohérentes pendant la recherche et nommées valeurs en échec (FV) peuvent être utilisées pour élaguer l'arbre de recherche par l'intermédiaire d'une nouvelle famille de cohérences locales qui est orthogonale aux cohérences habituelles. Etablir AFVC (la FV-cohérence d'arc) peut être réalisé avec une complexité relativement faible et permet de détecter de manière paresseuse une forme généralisée de la substituabilié de voisinage.

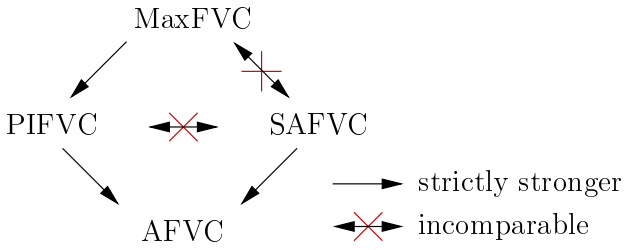

FIG. 8 – Relations entre les cohérences basées sur les valeurs en échec.

Bien qu'il soit cousin des approches qui éliminent les redondances en postant des contraintes [20, 15] ou en décomposant le problème [14], ce mode de raisonnement est différent car il définit une hiérarchie de cohérences de plus en plus fortes.

Pour les réseaux binaires, les valeurs en échec permettent d'identifier et d'exploiter une forme de nogood dans le même esprit que les nogoods généralisés de [16], les global cut seed de [10], les noyaux de [21] et les états partiels de [17]. Nous pensons que l'identification des propriétés communes de ces différentes approches est une perspective intéressante. Une autre perspective prometteuse est la mise au point d'algorithmes efficaces pour AFVC dans le cas n-aire.

Remerciements

Ce travail a bénéficié du soutien de l'IUT de Lens et du CNRS.

Références

[1] K.R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[2] A. Bellicha, C. Capelle, M. Habib, T. Kokény, and M.C. Vilarem. CSP techniques using partial orders on domain values. In *Proceedings of ECAI'94 workshop on constraint satisfaction issues raised by practical applications*, 1994.

[3] H. Bennaceur, C. Lecoutre, and O. Roussel. A decomposition technique for solving Max-CSP. In *Proceedings of ECAI'08*, pages 500–504, 2008.

[4] C. Bessiere. Constraint propagation. In *Handbook of Constraint Programming*, chapter 3. Elsevier, 2006.

[5] C. Bessiere, K. Stergiou, and T. Walsh. Domain filtering consistencies for non-binary constraints. *Artificial Intelligence*, 72(6-7) :800–822, 2008.

[6] D. Cohen, P. Jeavons, C. Jefferson, K.E. Petrie, and B. Smith. Symmetry definitions for constraint satisfaction problems. *Constraints*, 11(2-3) :115–137, 2006.

[7] M.C. Cooper. Fundamental properties of neighbourhood substitution in constraint satisfaction problems. *Artificial Intelligence*, 90 :1–24, 1997.

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

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

[10] F. Focacci and M. Milano. Global cut framework for removing symmetries. In *Proceedings of CP'01*, pages 77–92, 2001.

[11] E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32(4) :755–761, 1985.

[12] E.C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of AAAI'91*, pages 227–233, 1991.

[13] E.C. Freuder and C. Elfe. Neighborhood inverse consistency preprocessing. In *Proceedings of AAAI'96*, pages 202–208, 1996.

[14] E.C. Freuder and P.D. Hubbe. Using inferred disjunctive constraints to decompose constraint satisfaction problems. In *Proceedings of IJCAI'93*, pages 254–261, 1993.

[15] G. Gallo and G. Urbani. Algorithms for testing the satisfiability of propositional formulae. *Journal of Logic Programming*, 7(1) :45–61, 1989.

[16] G. Katsirelos and F. Bacchus. Generalized nogoods in CSPs. In *Proceedings of AAAI'05*, pages 390–396, 2005.

[17] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Exploiting past and future : Pruning by inconsistent partial state dominance. In *Proceedings of CP'07*, pages 453–467, 2007.

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

[19] U. Montanari. Network of constraints : Fundamental properties and applications to picture processing. *Information Science*, 7 :95–132, 1974.

[20] P.W. Purdom. Solving satisfiability with less searching. *IEEE transactions on pattern analysis and machine intelligence*, 6(4) :510–513, 1984.

[21] I. Razgon and A. Meisels. A CSP search algorithm with responsibility sets and kernels. *Constraints*, 12(2) :151–177, 2007.