Substituabilité au voisinage pour le cadre WCSP

Christophe Lecoutre, Djamel-Eddine Dehani, Olivier Roussel

Substituabilité au voisinage pour le cadre WCSP

Christophe Lecoutre Olivier Roussel Djamel E. Dehani

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

Résumé

WCSP est un problème d'optimisation pour lequel plusieurs formes de cohérences locales souples telles que, par exemple, la cohérence d'arc existentielle directionnelle (EDAC) et la cohérence d'arc virtuelle (VAC) ont été proposées durant ces dernières années. Dans cet article, nous adoptons une perspective différente en revisitant la propriété bien connue de la substituabilité (souple). Tout d'abord, nous précisons les relations existant entre la substituabilité de voisinage souple (SNS pour Soft Neighbourhood Substitutability) et une propriété appelée $pcost$ qui est basée sur le concept de surcoût de valeurs (par le biais de l'utilisation de paires de surcoût). Nous montrons que sous certaines hypothèses, $pcost$ est équivalent à SNS, mais que dans le cas général, elle est plus faible que SNS prouvée être coNP-difficile. Ensuite, nous montrons que SNS conserve la propriété VAC, mais pas la propriété EDAC. Enfin, nous introduisons un algorithme optimisé et nous montrons sur diverses séries d'instances WCSP l'intérêt pratique du maintien de $pcost$ avec AC*, FDAC ou EDAC, au cours de la recherche.

Abstract

WCSP is an optimization problem for which many forms of soft local (arc) consistencies such as existential directional arc consistency (EDAC) and virtual arc consistency (VAC) have been proposed these last years. In this paper, we adopt a different perspective by revisiting the well-known property of (soft) substitutability. First, we provide a clear picture of the relationships existing between soft neighborhood substitutability (SNS) and a tractable property called $pcost$ which is based on the concept of overcost of values (through the use of so-called cost pairs). We prove that under certain assumptions, $pcost$ is equivalent to SNS but weaker than SNS in the general case since we show that SNS is coNP-hard. We also show that SNS preserves the property VAC but not the property EDAC. Finally, we introduce an optimized algorithm and we show on various series of WCSP instances, the practical interest of maintaining $pcost$ together with AC*, FDAC or EDAC, during search.

1 Introduction

Le problème de satisfaction de contraintes valuées, VCSP pour Valued Constraint Satisfaction problem [23], est un cadre d'optimisation général utilisé avec succès pour manipuler le concept de contraintes souples dans de nombreuses applications en intelligence artificielle et en recherche opérationnelle. Une instance de problème VCSP est défini au moyen d'un ensemble de variables et d'un ensemble de fonctions de coût construites à partir d'une structure d'évaluation. Chaque fonction de coût détermine un degré de violation pour chaque instanciation possible d'un sous-ensemble de variables. Ces degrés (ou coûts) peuvent alors être combinés en utilisant l'opérateur $\oplus$ de la structure d'évaluation afin d'obtenir le coût global de toute instanciation complète. On peut globalement classer les différentes spécialisations du cadre VCSP selon les propriétés de l'opérateur $\oplus$: celles où $\oplus$ est idempotent (par exemple, min) et celles où $\oplus$ est (strictement) monotone (par exemple, $+$).

L'interchangeabilité est une propriété générale des réseaux de contraintes introduite par Freuder [13]. Deux valeurs a et b pour une variable x sont interchangeables si, pour toute solution I dans laquelle $x = b$, $I_{x=a}$ est également une solution, où $I_{x=a}$ désigne l'instanciation I dans laquelle a est assigné cette fois à x. L'interchangeabilité (complète) a été relaxée sous plusieurs formes telles que l'interchangeabilité de voisinage, la k-interchangeabilité, l'interchangeabilité partielle et la substituabilité. L'interchangeabilité et la substituabilité ont été

utilisées dans de nombreux contextes ; voir par exemple [2, 15, 6, 14, 1, 10, 22, 5, 18]. Une taxonomie partielle de ces deux propriétés peut être trouvée dans [16].

Une généralisation de la substituabilité pour les contraintes souples a été proposée dans [3] : une valeur a pour une variable x est susbtituable à une autre valeur b dans le domaine de x, si pour toute instanciation complète I impliquant (x, a), le coût de I est inférieur ou égal au coût de $I_{x=b}$. Il est particulièrement intéressant de noter que l'optimalité d'une instance est préservée lorsqu'on supprime une valeur pour laquelle une autre valeur est substituable. Calculer la substituabilité (complète) n'est pas traitable, mais la substituabilité de voisinage [3], qui est une forme limitée de la substituabilité où seules les contraintes impliquant une variable donnée sont considérées, peut être calculée en temps polynomial lorsque $\oplus$ est idempotent (à condition que l'arité des contraintes soit bornée). Cependant, lorsque $\oplus$ est monotone, comme c'est le cas pour la spécialisation VCSP appelée WCSP (Weighted CSP), aucun algorithme polynomial n'est connu.

Dans cet article, nous nous concentrons sur la substituabilité de voisinage souple (SNS) pour le cadre WCSP. Nous introduisons une propriété basée sur l'utilisation de paires de surcoût, appelée $pcost$, qui nous permet d'identifier efficacement des valeurs qui sont *souples-substituables*. Nous montrons que sous certaines hypothèses, $pcost$ est équivalent à SNS. Cependant, dans le cas général, lorsque k qui est la valeur qui représente le niveau des coûts interdits n'est pas ∞, $pcost$ est plus faible que SNS qui est par ailleurs démontré coNP-difficile. Nous étudions également les liaisons entre SNS et certaines propriétés connues de cohérence d'arc souple comme la cohérence d'arc existentielle directionnelle (EDAC) et la cohérence d'arc virtuelle (VAC). Nous démontrons que SNS préserve la propriété VAC, mais pas nécessairement la propriété EDAC. Enfin, nous développons un algorithme pour $pcost$ qui bénéficie d'une complexité en temps raisonnable, et nous montrons expérimentalement qu'il peut être associé à EDAC avec succès au cours de la recherche.

2 Contexte technique

Un *réseau de contraintes* (CN pour Constraint Network) P est un couple $(\mathscr{X}, \mathscr{C})$ où $\mathscr{X}$ est un ensemble fini de variables, noté par $vars(P)$, et $\mathscr{C}$ est un ensemble fini de contraintes. Chaque variable x possède un domaine (courant), noté $dom(x)$, qui est l'ensemble fini des valeurs qui peuvent être (couramment) affectées à x ; le domaine initial de x est noté $dom^{init}(x)$. d représente la taille du plus grand domaine. Chaque contrainte C_S implique un ensemble ordonné S de variables, appelés la *portée* de C_S, et représente une relation capturant l'ensemble des tuples autorisés pour les variables de S. Une contrainte *unaire* (resp., *binaire*) implique 1 (resp., 2) variable(s), et une contrainte *non-binaire* strictement plus que 2 variables. Une *instanciation* I d'un ensemble $X = \{x_1, \ldots, x_p\}$ de variables est un ensemble $\{(x_1, a_1), \ldots, (x_p, a_p)\}$ tel que $\forall i \in 1..p, a_i \in dom^{init}(x_i)$; X est noté $vars(I)$, et chaque a_i est noté par $I[x_i]$. Une instanciation I *sur* un CN P est une instanciation d'un ensemble $X \subseteq vars(P)$; elle est *complète* ssi $vars(I) = vars(P)$. I est *valide* sur P si et seulement si $\forall (x, a) \in I, a \in dom(x)$. I *couvre* une contrainte C_S ssi $S \subseteq vars(I)$. I *satisfait* une contrainte C_S avec $S = \{x_1, \ldots, x_r\}$ si et seulement si (i) I couvre C_S et (ii) le tuple $(I[x_1], \ldots, I[x_r]) \in C_S$. Une instanciation I sur un CN P est *localement cohérente* ssi (i) I est valide sur P et (ii) toutes les contraintes de P couvertes par I sont satisfaites par I. Une *solution* de P est une instanciation complète localement cohérente dans P.

Un *réseau de contraintes pondérées* (WCN pour Weighted CN) P est un triplet $(\mathscr{X}, \mathscr{C}, k)$ où $\mathscr{X}$ est un ensemble fini de n variables, $\mathscr{C}$ est un ensemble fini de e contraintes pondérées, également désigné par $cons(P)$, et $k > 0$ est un entier naturel ou $+\infty$. Chaque contrainte pondérée $c_S \in \mathscr{C}$ porte sur un ensemble ordonné S de variables (la portée), et est définie par une fonction de coût de $l(S)$ vers $\{0, \ldots, k\}$ où $l(S)$ est l'ensemble des instanciations possibles de S. Lorsque une instanciation a le coût k (noté aussi $\top$), elle est interdite. Sinon, elle est permise avec le coût correspondant (0, noté aussi par $\bot$, est entièrement satisfaisant). Afin de combiner les coûts, nous avons besoin de l'opérateur binaire $\oplus$ défini comme suit :

$$\forall a, b \in \{0, \ldots, k\}, a \oplus b = min(k, a + b)$$

Pour toute instanciation I et tout ensemble de variables X, soit $I_{\downarrow X} = \{(x, a) \mid (x, a) \in I \wedge x \in X\}$ la projection de I sur X. Nous désignons par $I_{x=a}$ l'instanciation $I_{\downarrow X \setminus \{x\}} \cup \{(x, a)\}$, qui est obtenue à partir de l'instanciation I soit par la substitution de la valeur affectée à x dans I par a, soit par l'extension de I avec (x, a). L'ensemble des contraintes voisines de x est désigné par $\Gamma(x) = \{c_S \in cons(P) \mid x \in S\}$. Quand $\Gamma(x)$ ne contient pas deux contraintes partageant au moins deux variables, on dit que $\Gamma(x)$ est *séparable*. Si c_S est une contrainte (pondérée) et I est une instanciation d'un ensemble $X \supseteq S$, alors $c_S(I)$ sera considéré comme égal à $c_S(I')$ où I' est la restriction de I aux variables de S (en d'autres termes, les projections seront implicites). Si C est un ensemble de contraintes, alors $vars(C) = \cup_{c_S \in C} S$ est l'ensemble des variables impliquées dans C ; si I est une instanciation telle que $vars(C) \subseteq vars(I)$, alors $cost_C(I) = \bigoplus_{c_S \in C} c_S(I)$ est le coût de I obtenu en considérant toutes les contraintes de C. Pour un WCN P et une instanciation complète I de P, le coût de I est alors $cost_{cons(P)}(I)$ qui sera simplifié en $cost(I)$. La tâche usuelle (NP-difficile) du problème de satisfaction de contraintes pondérées (WCSP) est de trouver, pour un WCN donné, une instanciation complète dont le coût est minimal. CSP peut être considéré comme une spécialisation de WCSP (lorsque seulement les coûts 0 et

k sont utilisés) et WCSP [20] peut être considéré comme une spécialisation du cadre générique de contraintes valuées [4].

De nombreuses formes de cohérence d'arc souple ont été proposées au cours des dernières années (voir par ex. [9]). Nous allons brièvement les introduire dans le contexte des WCNs binaires. Sans perte de généralité, on supposera l'existence d'une contrainte d'arité zéro notée $c_\emptyset$ (une constante) ainsi que la présence d'une contrainte unaire c_x pour chaque variable x. Une variable x est noeud-cohérente (NC*) si et seulement si $\forall a \in dom(x), c_\emptyset \oplus c_x(a) < k$ et $\exists b \in dom(x) \mid c_x(b) = 0$. Une variable x est arc-cohérente (AC*) si et seulement si x est NC* et $\forall a \in dom(x), \forall c_{xy} \in cons(P), \exists b \in dom(y) \mid c_{xy}(a,b) = 0$; b est appelé un support simple de a. Un WCN est AC^* ssi chacune de ses variables est AC^* [19, 20]. Un WCN est arc-cohérent directionnel complet (FDAC) [7] par rapport à un ordre total $<$ sur les variables ssi il est AC^* et $\forall c_{xy} \mid x < y, \forall a \in dom(x), \exists b \in dom(y) \mid c_{xy}(a,b) = c_y(b) = 0$; b est appelé un support complet de a. Un WCN est arc-cohérent existentiel (EAC) [12] ssi il est NC* et $\forall x \in vars(P), \exists a \in dom(x) \mid c_x(a) = 0 \wedge \forall c_{xy}, \exists b \in dom(y) \mid c_{xy}(a,b) = c_y(b) = 0$ (a est appelé le support existentiel de x) . Un WCN est arc-cohérent directionnel existentiel (EDAC) par rapport à un ordre $<$ sur les variables ssi il est EAC et FDAC par rapport à $<$. On notera $\phi(P)$ l'établissement de la propriété ϕ (par exemple AC ou EDAC) sur le (W)CN P. Pour tout WCN P, nous pouvons construire un CN noté $Bool(P)$ qui est obtenu en transformant chaque contrainte souple en une contrainte (dure) où seuls les tuples avec un coût nul sont autorisés. P est virtuel arc-cohérent (VAC) [8] ssi $AC(Bool(P)) \neq \bot$. Un WCN est optimal arc-cohérent souple (OSAC) ssi aucune transformation EPT (voir [9]) qui peut lui être appliquée ne permet d'augmenter $c_\emptyset$. OSAC est plus fort que VAC, qui lui-même est plus fort que EDAC, lorsqu'on compare les valeurs de $c_\emptyset$.

3 La substituabilité souple

Dans cette section, nous introduisons la substituabilité souple (au voisinage) [13, 3]. À partir de maintenant, nous considérons un (W)CN P donné.

Définition 1 *Soient $x \in vars(P)$ et $\{a,b\} \subseteq dom(x)$. (x,a) est* souple-substituable *à (x,b) dans P ssi pour toute instanciation complète I de P, $cost(I_{x=a}) \leq cost(I_{x=b})$.*

Lorsque (x,a) est souple-substituable à (x,b), b peut être supprimé de $dom(x)$ sans changer le coût optimum de P. En effet, si elles existent, les solutions optimales possibles de P avec (x,b) sont perdues, mais il est garanti qu'il reste au moins une solution optimale avec (x,a).

Parce qu'elle implique de traiter toutes les instanciations complètes, l'identification des valeurs souples-substituables est inutilisable en pratique. Cependant, il y a une forme de substituabilité locale, appelé substituabilité au voisinage [13, 3] qui peut être utile. Pour le cadre WCSP, elle est définie comme suit :

Définition 2 *Soient $x \in vars(P)$ et $\{a,b\} \subseteq dom(x)$, (x,a) est souple-substituable à (x,b)* au voisinage *dans P ssi pour chaque instanciation complète I de P, $cost_{\Gamma(x)}(I_{x=a}) \leq cost_{\Gamma(x)}(I_{x=b})$.*

On dira que (x,b) est SNS-éliminable (dans P) quand il existe une valeur (x,a) souple-substituable à (x,b) au voisinage. La substituabilité souple au voisinage implique la substituabilité souple (complète) (mais l'inverse n'est pas vrai). Il est particulièrement intéressant de noter que la substituabilité souple au voisinage permet une compensation de coûts entre contraintes souples. Une telle compensation est rendue possible par la présence de tous les coûts intermédiaires dans la structure d'évaluation, c'est-à-dire, les coûts différents de 0 et de k Par exemple, considérons un WCN composé de trois variables x, y et z avec $dom(x) = dom(y) = dom(z) = \{a,b\}$ et deux contraintes pondérées c_{xy} et c_{xz} telles que $c_{xy}(a,a) = c_{xz}(b,a) = c_{xz}(b,b) = 1$; tous les autres coûts étant à 0. Une illustration de ce WCN est donnée par la figure 1 ; les contraintes binaires sont représentées avec des arêtes étiquetées, et les coûts nuls ne sont pas représentés. Notons que (x,a) est souple-substituable à (x,b) au voisinage grâce à la compensation des coûts.

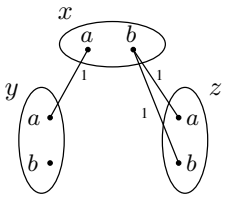


FIGURE 1 – (x,a) est souple-substituable à (x,b) au voisinage

La définition 2 nécessite de considérer chaque instanciation de $vars(\Gamma(x))$, ce qui reste très coûteux. La considération de chaque contrainte prise individuellement permettrait de réduire ce coût, mais dans ce cas, il est nécessaire de pouvoir identifier les compensations de coûts entre les différentes contraintes. Une façon simple de le faire est de calculer une somme de surcoûts minimaux, c'est à dire, une somme de différences de coût minimales sur toutes les contraintes : $ocost(x : a \rightarrow b) = \sum_{c_S \in \Gamma(x)} min_{I \in l(S)} \{c_S(I_{x=b}) - c_S(I_{x=a})\}$, tel que mentionné dans [17, 11]. Malheureusement, cette soustraction de coûts pose des problèmes subtils quand $k \neq \infty$. Ceci est illustré ci-dessous.

Exemple 1 Considérons les deux familles de contraintes $C_i = \{c_i \mid i \in 1..n\}$ et $C'_i = \{c'_i \mid i \in 1..n'\}$ définis par :

x	y_i	c_i
a	c	0
b	c	1

x	z_i	c'_i
a	d	1
b	d	0

Quand $n = k$ et $n' = k + 1$, (x, a) et (x, b) sont interchangeable parce que les deux valeurs sont interdites. Cependant, $\forall c_i \in C_i, cost_{c_i}(I_{x=b}) - cost_{c_i}(I_{x=a}) = 1$ et $\forall c'_i \in C'_i, cost_{c'_i}(I_{x=b}) - cost_{c'_i}(I_{x=a}) = -1$. En considérant la somme des différences sur les contraintes de $\Gamma(x) = C_i \cup C'_i$, la valeur résultante est -1 ce qui indiquerait que b a globalement un coût inférieur à celui a, ce qui est faux puisque les deux valeurs a et b ont un coût de k. Pour identifier correctement ce cas de substituabilité souple lorsque $k \neq \infty$, il est obligatoire d'utiliser un opérateur non commutatif, qui nous empêche d'utiliser l'opérateur $-$ usuel.

4 Calcul de la substituabilité souple

Nous allons maintenant nous concentrer sur la substituabilité souple au voisinage, et plus précisément sur la complexité de l'identification des valeurs SNS-éliminables. Nous commençons par discuter de certains travaux connexes à notre démarche. Pour le cadre général VCSP, des algorithmes efficaces pour la substituabilité souple au voisinage existent [3] lorsque l'opérateur d'agrégation de la structure d'évaluation VCSP est idempotent. Pour le cadre FCSP (Fuzzy CSP), la notion de substituabilité floue au voisinage est proposée dans [7] : il est montré que les valeurs floues-substituables au voisinage peuvent être efficacement identifiées lorsque l'opérateur d'agrégation de la structure d'évaluation FCSP est strictement monotone ou lorsque il s'agit de l'opérateur max. Plus récemment, la possibilité de calculer des formes de dominance plus faibles que la substituabilité souple au voisinage a été proposé dans [17]. Cependant, aucune étude précise qualitative n'a été menée pour le cadre WCSP. C'est ce que nous proposons de faire maintenant.

Tout d'abord, nous introduisons les paires de surcoût sur lesquelles notre mécanisme de calcul se base (ceci peut aussi être relié à ce qui a été proposé dans [7] pour le cadre FCSP). En effet, une manière de contourner les problèmes de soustraction mentionnés ci-dessus est d'utiliser seulement l'addition sur des paires de surcoût. Cette méthode est analogue à la construction d'entiers comme classes d'équivalence de paires ordonnées de nombres naturels où une paire (β, α) représente l'entier $\beta - \alpha$. Nous définissons $+$ (addition) sur les paires de surcoût par $(\beta, \alpha) + (\beta', \alpha') = (\beta + \beta', \alpha + \alpha')$ (ceci est le $+$ usuel et non $\oplus$) et la comparaison des paires de surcoût avec 0 par $(\beta, \alpha) \geq 0 \Leftrightarrow \beta \geq \alpha$. Les paires sont ordonnées par la relation $\leq$ définie par $(\beta, \alpha) \leq (\beta', \alpha') \Leftrightarrow (\beta - \alpha < \beta' - \alpha') \vee (\beta - \alpha = \beta' - \alpha' \wedge \alpha < \alpha')$. En un sens, la paire (β, α) porte deux informations : la différence $\beta - \alpha$ mais aussi $min(\beta, \alpha)$ qui est perdu lorsqu'on utilise une simple soustraction.

Définition 3 *Soient $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$,*
- *la* paire de surcoût *de (x, b) vis à vis de (x, a) dans $c_S \in \Gamma(x)$ est définie par $pcost(c_S, x : a \to b) = min_{I \in l(S)}\{(c_S(I_{x=b}), c_S(I_{x=a}))\}$;*
- *la* paire de surcoût *de (x, b) vis à vis de (x, a) dans P est définie par $pcost(x : a \to b) = \sum_{c_S \in \Gamma(x)} pcost(c_S, x : a \to b)$.*

Proposition 1 *Soient $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$. Si $pcost(x : a \to b) \geq 0$ alors (x, a) est souple-substituable à (x, b) au voisinage dans P.*

Preuve. Pour une contrainte c_S, soit I^{c_s} l'instanciation de $S - \{x\}$ qui donne la paire de surcoût minimale dans $min_{I \in l(S)}\{(c_S(I_{x=b}), c_S(I_{x=a}))\}$. Par définition, $pcost(c_S, x : a \to b) = (c_S(I^{c_S}_{x=b}), c_S(I^{c_S}_{x=a}))$. Par définition de min sur les paires de surcoût, nous avons $\forall I, \forall c_S \in \Gamma(x)$, $(c_S(I_{x=b}), c_S(I_{x=a})) \geq (c_S(I^{c_S}_{x=b}), c_S(I^{c_S}_{x=a}))$. Par sommation, nous obtenons $\forall I$, $\sum_{c_S \in \Gamma(x)}(c_S(I_{x=b}), c_S(I_{x=a})) \geq \sum_{c_S \in \Gamma(x)}(c_S(I^{c_S}_{x=b}), c_S(I^{c_S}_{x=a}))$. Par hypothèse, $pcost(x : a \to b) \geq 0$, donc nous avons $\sum_{c_S \in \Gamma(x)}(c_S(I^{c_S}_{x=b}), c_S(I^{c_S}_{x=a})) \geq 0$, et donc $\forall I, \sum_{c_S \in \Gamma(x)}(c_S(I_{x=b}), c_S(I_{x=a})) \geq 0$. Par définitions de $+$ et $\leq$ sur les paires de surcoût, nous pouvons tirer $\forall I, \sum_{c_S \in \Gamma(x)} c_S(I_{x=b}) \geq \sum_{c_S \in \Gamma(x)} c_S(I_{x=a})$ ce qui implique $\forall I, min(k, \sum_{c_S \in \Gamma(x)} c_S(I_{x=b})) \geq min(k, \sum_{c_S \in \Gamma(x)} c_S(I_{x=a}))$. Puisque $\forall a_i \in \{0, \ldots, k\}$, $a_1 \oplus \ldots \oplus a_n = min(k, a_1 + \ldots a_n\}$, nous pouvons conclure que $\forall I, \bigoplus_{c_S \in \Gamma(x)} c_S(I_{x=b})) \geq \bigoplus_{c_S \in \Gamma(x)} c_S(I_{x=a}))$. Ainsi, (x, a) est souple-substituable à (x, b) au voisinage dans P. □

La réciproque de la proposition 1 n'est pas vraie dans le cas général. Un premier cas où elle est fausse apparaît lorsque les portées de deux contraintes non binaires ont une intersection de plus d'une variable (voisinage non séparable). Dans cette situation, les contraintes ne peuvent pas être considérées individuellement.

Exemple 2 Soient quatre variables x, y, z et t telles que $dom(x) = \{a, b\}, dom(y) = \{c, d\}, dom(z) = dom(t) = \{e\}$, et deux contraintes ternaires c_{xyz}, c_{xyt} définies par la table de coûts suivante :

x	y	z	t	c_{xyz}	c_{xyt}
a	c	e	e	1	0
b	c	e	e	0	1
a	d	e	e	0	1
b	d	e	e	1	0

Il est assez facile de constater que (x, a) est souple-substituable à (x, b) au voisinage alors que $pcost(c_{xyz}, x, a \to b) = pcost(c_{xyt}, x, a \to b) = (0, 1)$ et, par conséquent $pcost(x, a \to b) = (0, 2) \not\geq 0$.

Ainsi, une première condition pour que la réciproque de la proposition 1 tienne est que $\Gamma(x)$ soit séparable (ce qui est le cas des réseaux binaires normalisés). Un autre cas de figure où la réciproque de la proposition 1 est fausse est quand $k \neq \infty$. Considérant le WCN de l'exemple 1 avec $n = k$ et $n' = k + 1$, nous pouvons observer que $pcost(x : a \to b) = (n, n') = (k, k + 1) \not\geq 0$. Cependant, (x, a) et (x, b) sont tous deux interdits et par conséquent (x, a) est souple-substituable à (x, b) (et inversement). Dans cet exemple, il pourrait sembler une bonne idée d'utiliser $\oplus$ au lieu de $+$ pour l'addition de paires. Cependant, l'exemple 3 montre que cela conduirait à l'identification erronée de valeurs substituables.

Exemple 3 Considérons la contrainte unaire c_x et la famille de contraintes binaires $C_i = \{c_i \mid i \in 1..n\}$ définie par :

x	y_i	c_i
a	a	2
a	b	0
b	a	1
b	b	0

x	c_x
a	1
b	0

Clairement, (x, a) est non substituable à (x, b). Posons $n = k$. Avec la somme des paires définie par $+$, $pcost(x, a \to b) = (n, 2n + 1) \not\geq 0$. Si la somme de paires est définie par $\oplus$, nous obtiendrions $(k, k) \geq 0$.

Heureusement, il y a des situations où, même lorsque $k \neq \infty$, l'utilisation des paires de surcoût nous permet d'identifier précisément l'ensemble des valeurs souples-substituables au voisinage.

Proposition 2 *Soient $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$ tel que $\Gamma(x)$ est séparable et $pcost(x, a \to b) = (\beta, \alpha)$ avec $\alpha < k$. Si (x, a) est souple-substituable à (x, b) au voisinage dans P alors $pcost(x : a \to b) \geq 0$.*

Preuve. Puisque par hypothèse $\Gamma(x)$ est séparable, nous pouvons définir l'instanciation I^{min} dans $vars(\Gamma(x)) \setminus \{x\}$ comme l'union pour chaque contrainte $c_s \in \Gamma(x)$ de l'instanciation I^{c_S} définie dans la preuve de la proposition 1. I^{min} est tel que $pcost(c_S, x : a \to b) = (c_S(I^{min}_{x=b}), c_S(I^{min}_{x=a}))$.

Par hypothèse, $\forall I, cost_{\Gamma(x)}(I_{x=b}) \geq cost_{\Gamma(x)}(I_{x=a})$, ce qui peut être réécrit sous la forme $\forall I, \bigoplus_{c_S \in \Gamma(x)} c_S(I_{x=b}) \geq \bigoplus_{c_S \in \Gamma(x)} c_S(I_{x=a})$. En particulier, cela est vrai pour $I = I^{min}$. Par conséquent, $\bigoplus_{c_S \in \Gamma(x)} c_S(I^{min}_{x=b}) \geq \bigoplus_{c_S \in \Gamma(x)} c_S(I^{min}_{x=a})$ qui peut être réécrit sous la forme $min(k, \beta) \geq min(k, \alpha)$ où $\beta = \sum_{c_S \in \Gamma(x)} c_S(I^{min}_{x=b}))$ et $\alpha = \sum_{c_S \in \Gamma(x)} c_S(I^{min}_{x=a})$. Par définition, $pcost(x : a \to b) = (\beta, \alpha)$ et donc $pcost(x : a \to b) \geq 0$ ssi $\beta \geq \alpha$. Maintenant, si $\alpha < k$, $min(k, \alpha) = \alpha$ et $min(k, \beta) \geq min(k, \alpha) \Rightarrow \beta \geq \alpha \Rightarrow pcost(x : a \to b) \geq 0$ (ceci est vrai pour $\beta < k$ et $\beta \geq k$). Notons que lorsque $\alpha \geq k$, $min(k, \beta) \geq min(k, \alpha) \not\Rightarrow \beta \geq \alpha$; un contre-exemple étant $\beta = k$ et $\alpha = k + 1$. □

Corollaire 1 *Soient $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$ tel que $\Gamma(x)$ est séparable et $pcost(x : a \to b) = (\beta, \alpha)$ avec $\alpha < k$. Si $(\beta, \alpha) < 0$ alors (x, a) n'est pas souple-substituable à (x, b) au voisinage dans P.*

Quand $pcost(x : a \to b) = (\beta, \alpha)$ avec $\alpha \geq k$, décider si (x, a) est souple-substituable à (x, b) au voisinage est beaucoup plus difficile. En effet, ce problème est coNP difficile. Pour prouver cela, nous introduisons le problème de double coût à choix multiple.

Le problème de choix multiple à double coût (MCDCP) Étant donnés m ensembles E_1, E_2, ..., E_m d'objets tels que chaque objet $o_j \in E_i$ a une valeur de coût principale $r_{ij} \in \mathbb{Z}^+$ ainsi qu'une valeur de coût secondaire $s_{ij} \in \mathbb{Z}^+$. Étant donné un coût maximal $C \in \mathbb{Z}^+$, le problème MCDCP consiste à décider si il est possible de choisir un objet dans chaque ensemble de telle sorte que la somme des coûts principaux de ces objets sélectionnés ne dépasse pas C et ne dépasse pas également la somme des coûts secondaires. Ce problème peut être formulé comme suit :

$$\begin{array}{l} \sum_{i=1}^{m} \sum_{j \in E_i} r_{ij} x_{ij} \leq C, \\ \sum_{i=1}^{m} \sum_{j \in E_i} r_{ij} x_{ij} \leq \sum_{i=1}^{m} \sum_{j \in E_i} s_{ij} x_{ij}, \\ \sum_{j \in E_i} x_{ij} = 1, i = 1, \ldots, m, \\ x_{ij} \in \{0, 1\}, i = 1, \ldots, m, j \in E_i. \end{array}$$

Proposition 3 *Le problème de choix multiple à double coût est NP-complet.*

Preuve. L'adhésion à NP est immédiat. Pour la NP-difficulté, nous réduisons le problème de sac à dos à choix multiple (MCKP pour Multiple-Choice Knapsack Problem) au problème de choix multiple à double coût. MCKP est connu pour être NP-difficile (par exemple, voir [21]). Pour MCKP, nous avons aussi m ensembles, et un profit $p_{ij} \in \mathbb{Z}^+$ est associé à chaque objet ainsi qu'un poids $w_{ij} \in \mathbb{Z}^+$. Étant donné un bénéfice minimal $P \in \mathbb{Z}^+$ et un poids maximal $W \in \mathbb{Z}^+$, le problème de décision MCKP est formulé comme suit :

$$\begin{array}{l} \sum_{i=1}^{m} \sum_{j \in E_i} w_{ij} x_{ij} \leq W, \\ \sum_{i=1}^{m} \sum_{j \in E_i} p_{ij} x_{ij} \geq P, \\ \sum_{j \in E_i} x_{ij} = 1, i = 1, \ldots, m, \\ x_{ij} \in \{0, 1\}, i = 1, \ldots, m, j \in E_i. \end{array}$$

Pour coder une instance MCKP en une instance MCDCP, nous conservons la même structure (ensembles) et définissons :

$C = qW - mP$,
$r_{ij} = qw_{ij} - P, i = 1, \ldots, m, j \in E_i$,
$s_{ij} = mp_{ij} + r_{ij} - P, i = 1, \ldots, m, j \in E_i$,

avec $q = 2mP$. Avec cette valeur choisie pour q, on peut montrer que toutes les valeurs C, r_{ij} et s_{ij} appartiennent à $\mathbb{Z}^+$. La première équation MCDCP peut être transformée comme suit : $\sum_{i=1}^{m} \sum_{j \in E_i} r_{ij} x_{ij} \leq C$
$\Rightarrow \sum_{i=1}^{m} \sum_{j \in E_i} (qw_{ij} - P) x_{ij} \leq qW - mP$
$\Rightarrow \sum_{i=1}^{m} \sum_{j \in E_i} qw_{ij} x_{ij} - mP \leq qW - mP$ car exactement m variables x_{ij} sont assignées à 1,
$\Rightarrow \sum_{i=1}^{m} \sum_{j \in E_i} w_{ij} x_{ij} \leq W$.
La seconde équation MCDCP peut être transformée comme suit : $\sum_{i=1}^{m} \sum_{j \in E_i} r_{ij} x_{ij} \leq \sum_{i=1}^{m} \sum_{j \in E_i} s_{ij} x_{ij}$,
$\Rightarrow 0 \leq \sum_{i=1}^{m} \sum_{j \in E_i} (mp_{ij} - P) x_{ij}$ en simplifiant r_{ij} des deux côtés,
$\Rightarrow 0 \leq \sum_{i=1}^{m} \sum_{j \in E_i} mp_{ij} x_{ij} - mP$ car exactement m variables x_{ij} sont assignées à 1,
$\Rightarrow P \leq \sum_{i=1}^{m} \sum_{j \in E_i} p_{ij} x_{ij}$. □

Proposition 4 *Décider si une valeur est souple-substituable au voisinage à une autre dans un WCN $(\mathscr{X}, \mathscr{C}, k)$ où $k \neq \infty$ est coNP-difficile.*

Preuve. Toute instance MCDCP peut être réduit au problème de décider si une valeur (x, a) n'est pas souple-substituable à une autre valeur (x, b) au voisinage dans un WCN P. A partir de l'instance MCDCP, nous construisons le WCN P comme suit :

- $vars(P)$ contient une variable x tel que $dom(x) = \{a, b\}$ et une variable y_i pour chaque ensemble E_i ; le domaine de y_i contient les objects o_{i1}, o_{i2}, ... de E_i.
- $cons(P)$ contient exactement m contraintes binaires souple c_{xy_i} : nous avons $c_{xy_i}(\{(x, a), (y_i, o_{ij})\}) = s_{ij}$ et $c_{xy_i}(\{(x, b), (y_i, o_{ij})\}) = r_{ij}$.
- k est fixé à C.

Déterminer si (x, a) n'est pas souple-substituable à (x, b) au voisinage dans P est équivalent à trouver une instanciation I de $vars(\Gamma(x))$ telle que $cost_{\Gamma(x)}(I_{x=a}) > cost_{\Gamma(x)}(I_{x=b})$ qui est équivalent à $\sum_{c_S \in \Gamma(x)} c_s(I_{x=b}) < k \wedge \sum_{c_S \in \Gamma(x)} c_s(I_{x=a}) > \sum_{c_S \in \Gamma(x)} c_s(I_{x=b})$. La première (resp. seconde) condition encode la première (resp. seconde) inégalité de l'instance MCDCP. Comme les variables x_{ij} de l'instance MCDCP correspondent à l'affectation des variables y_i dans le WCN ($x_{ij} = 1 \Leftrightarrow y_i = o_{ij}$), la troisième équation de l'instance MCDCP est directement prise en compte dans le WCN. □

En pratique, il est plus facile de manipuler des différences de coûts en utilisant $ocost(x : a \rightarrow b) = \sum_{c_S \in \Gamma(x)} min_{I \in l(S)} \{c_S(I_{x=b}) - c_S(I_{x=a})\}$. En un sens, $pcost$ est plus précis que $ocost$: quand $pcost(x : a \rightarrow b) = (\beta, \alpha)$ avec $\alpha < k$, (x, a) souple-substituable à (x, b) au voisinage est équivalent à $\beta \geq \alpha$, et quand $\alpha \geq k$, aucune conclusion ne peut être donnée. Avec $ocost$, l'information α est perdue. Par conséquent, lorsque $ocost(x : a \rightarrow b) < 0$, nous ne pouvons pas conclure avec certitude que (x, a) n'est pas souple-substituable à (x, b) au voisinage. Dans la pratique, cela fait peu de différence (tout du moins, si on considère nos développements algorithmiques actuels), ce qui est la raison pour laquelle les expériences dans le présent document ont été réalisées avec $ocost$.

5 Liaisons avec la cohérence d'arc souple

Après l'introduction de la fermeture par substituabilité souple au voisinage, cette section présente quelques résultats connectant la substituabilité souple au voisinage avec diverses formes de cohérence d'arc souple.

Définition 4 *La fermeture par substituabilité souple au voisinage (ou SNS-closure) d'un WCN P, noté $SNS(P)$, est tout WCN obtenu après l'élimination itérative des valeurs SNS-éliminables jusqu'à ce qu'un point fixe soit atteint.*

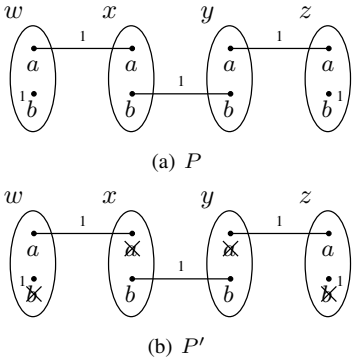
Puisque cette opération n'est pas confluente, $SNS(P)$ n'est pas unique. Quand nous utilisons l'approche $pcost$ pour identifier des valeurs SNS-éliminables, on notera $PSNS(P)$.

Proposition 5 *Soit P un WCN EDAC-cohérent. $SNS(P)$ n'est pas nécessairement EDAC-cohérent.*

Preuve. Considérons le WCN P représenté par la figure 2(a). Notons que P est EDAC-cohérent par rapport à l'ordre $w > x > z > y$, et que (w, a) et (z, a) sont respectivement souples-substituables à (w, b) et (z, b) au voisinage, puisque $pcost(w, a \rightarrow b) = (0, 0)$ et $pcost(z, a \rightarrow b) = (0, 0)$. Il existe une SNS-closure unique de P, $P' = SNS(P)$, qui est représentée par la figure 2(b). Il est clair que P', n'est pas EDAC-cohérent puisque (x, b) et (y, b) n'ont pas de support sur c_{xy}. □

Lemma 1 *Soient P un WCN VAC-cohérent, $x \in S$ et $\{a, b\} \subseteq dom(x)$ tel que (x, b) est AC-cohérent dans $Bool(P)$. Si $pcost(x : a \rightarrow b) \geq 0$ dans P alors (x, a) est substituable (au sens CSP [13]) à (x, b) au voisinage dans $Bool(P)$.*

Preuve. Nous supposons que P est VAC-cohérent et (x, b) est AC-cohérent dans $Bool(P)$. Puisque (x, b) est AC-cohérent dans $Bool(P)$, nous savons que pour chaque contrainte $c_S \in \Gamma(x)$, il existe une instanciation I de S telle que $I[x] = b$ et $c_S(I) = 0$ (par construction de $Bool(P)$). Cela veut dire que pour chaque contrainte



(a) P

(b) P'

FIGURE 2 – EDAC versus SNS

$c_S \in \Gamma(x)$, $pcost(c_S, x : a \to b) \leq 0$. Comme par hypothèse $pcost(x : a \to b) \geq 0$, pour chaque $c_S \in \Gamma(x)$, nous avons nécessairement $pcost(c_S, x : a \to b) = 0$. Nous pouvons déduire que pour chaque contrainte $c_S \in \Gamma(x)$, pour chaque instanciation I de S telle que $I[x] = b$ et $c_S(I) = 0$, l'instanciation $I' = I_{x=a}$ est telle que $c_S(I') = 0$. Pour finir, nous pouvons déduire que (x, a) est substituable à (x, b) au voisinage dans $Bool(P)$. □

Proposition 6 *Soient P un WCN VAC-cohérent, $x \in vars(P)$ et $\{a, b\} \subseteq dom(x)$. Si $pcost(x : a \to b) \geq 0$ dans P alors $P \setminus \{(x, b)\}$ est VAC-cohérent.*

Preuve. Supposons (premier cas) pour commencer que (x, b) est AC-cohérent dans $Bool(P)$. D'après le lemme précédent, nous savons que (x, a) est substituable à (x, b) au voisinage dans $Bool(P)$, et donc nous avons $AC(Bool(P)) \neq \bot \Leftrightarrow AC(Bool(P \setminus \{(x, b)\})) \neq \bot$. Puisque P est VAC-cohérent, nécessairement $P \setminus \{(x, b)\}$ est VAC-cohérent. Maintenant (deuxième cas), supposons que (x, b) n'est pas AC-cohérent dans $Bool(P)$. Il est clair que nous avons alors $AC(Bool(P)) = AC(Bool(P \setminus \{(x, b)\}))$, et donc $P \setminus \{(x, b)\}$ est VAC-cohérent (puisque P est VAC-cohérent). □

Corollaire 2 *Soit P un WCN. Si P est VAC-cohérent alors $SNS(P)$ est VAC-cohérent.*

Le corollaire précédent est également valable pour OSAC.

6 Algorithme

Dans cette section, nous présentons un algorithme pour établir AC*+PSNS (qui peut être facilement adapté à EDAC*+PSNS, par exemple). L'idée directrice est de toujours commencer à identifier les valeurs SNS-éliminables à partir d'un WCN qui est AC*-cohérent. Cela nous permet de réduire l'effort de calcul en sortant de boucles avant

Algorithme 1: overcost($c_S, x : a \to b$) : entier

```
ocst ← 1
foreach I ∈ l(S \ {x}) do
    if c_S(I_{x=b}) − c_S(I_{x=a}) < ocst then
        ocst ← c_S(I_{x=b}) − c_S(I_{x=a})
return ocst
```

Algorithme 2: overcost($x, a \to b$) : entier

```
ocst ← c_x(b) − c_x(a)
if ocst < 0 then
    return ocst
ocst ← ocst + overcost(residues[x, a, b], x, a → b)
if ocst < 0 then
    return ocst
foreach c_S ∈ Γ(x) | c_S ≠ residues[x, a, b] do
    d ← overcost(c_S, x, a → b)
    if d < −(c_x(b) − c_x(a)) then
        residues[x, a, b] ← c_S
    ocst ← ocst + d
    if ocst < 0 then
        return ocst
return ocst
```

Algorithme 3: PSNS^r(P : WCN AC*-consistent)

```
Δ ← ∅
foreach x ∈ vars(P) do
    if ∃y ∈ Γ(x) | stamp[y] > substamp then
        foreach (a, b) ∈ dom(x)^2 | b > a do
            if overcost(x, a → b) then
                Δ ← Δ ∪ {(x, b)}
            else if overcost(x, b → a) then
                Δ ← Δ ∪ {(x, a)}
substamp ← time++
foreach (x, a) ∈ Δ do
    remove (x, a) from dom(x)
    Q ← Q ∪ {x}
    stamp[x] ← time++
```

Algorithme 4: AC*-PSNS(P : WCN)

Output: P, rendu AC*-consistent et PSNS-clos

```
time ← 0
substamp ← −1
stamp[x] ← 0, ∀x ∈ vars(P)
Q ← vars(P)
repeat
    PSNS^r(W-AC*(P, Q))
until Q ≠ ∅
```

leur fin et en utilisant des résidus. Pour simplifier, les valeurs SNS-éliminables sont identifiées à l'aide des surcoûts $ocost$ parce que, comme mentionné plus haut, $pcost$ et $ocost$ donnent les mêmes réponses positives ($pcost$ n'étant plus précis que pour les réponses négatives que l'on n'exploite pas ici).

La procédure principale est l'algorithme 4. Comme souvent, nous utilisons un ensemble, noté Q, pour stocker les variables dont le domaine a été récemment réduit. Au début, Q contient toutes les variables (ligne 4). Puis, à la ligne 6, un algorithme classique AC*, désigné ici par W-AC*, est exécuté (par exemple, cela peut être W-AC*2001 [20]), avant de solliciter une fonction appelée PSNS^r. Les appels de W-AC* et de PSNS^r sont entrelacés jusqu'à ce qu'un point fixe soit atteint (i.e., $Q = \emptyset$).

La fonction PSNS^r, algorithme 3, itère sur toutes les variables afin de recueillir les valeurs SNS-éliminable dans un ensemble appelé Δ. Cet ensemble est initialisé à la ligne 1 et mis à jour aux lignes 6 et 8. Imaginons que toutes les valeurs SNS-éliminables (qui peuvent être identifiées avec le calcul de surcoûts) pour une variable x ont été supprimées, et que le domaine de toutes les variables dans le voisinage de x reste identique. Clairement, il n'est alors pas nécessaire de considérer à nouveau x pour rechercher des valeurs SNS-éliminables. C'est l'objet de la ligne 3. Ici, un mécanisme d'horodatage est utilisé. En introduisant un compteur global $time$ et en associant un tampon temporel $stamp[x]$ à chaque variable x ainsi qu'un tampon $substamp$ à la fonction PSNS^r, il est possible de déterminer quelles variables doivent être considérées. La valeur de $stamp[x]$ indique à quel moment une valeur a été récemment éliminée de $dom(x)$ tandis que la valeur de $substamp$ indique à quel moment PSNS^r a été récemment appelé. Les variables $time$, $stamp[x]$ pour chaque variable x et $substamp$ sont initialisées aux ligne 1 à 3 de l'algorithme 4. La valeur de $time$ est incrémentée chaque fois qu'une variable est ajoutée à Q (ligne 13 de l'algorithme 3, et cela *doit* aussi être effectué au sein de W-AC*) et chaque fois que PSNS^r est appelé (ligne 9). Toutes les valeurs SNS-éliminables collectées dans Δ sont supprimées tandis que Q est mis à jour pour pour le prochain appel à W-AC* (lignes 10 à 12).

L'algorithme 2 nous permet de calculer le surcoût $ocost$ de (x, b) par rapport à (x, a). Puisque nous savons que le WCN est AC*-cohérent, nous avons la garantie que le surcoût de (x, b) par rapport à (x, a) dans toute contrainte non-unaire c_S où $x \in S$ est inférieur où égal à 0. Cela signifie que nous ne pourrons jamais compenser une valeur négative avec une valeur positive (une fois que le surcoût unaire a été pris en compte). Un grand avantage de cette observation est la possibilité d'utiliser les arrêts précoces de boucle au cours de tels calculs. Cette opération est effectuée aux lignes 3, 6 et 13. Les résidus sont un autre mécanisme introduit pour augmenter la performance de l'algorithme. Pour chaque variable x, et pour tout couple (a, b) de valeurs de $dom(x)$, nous stockons dans $residues[x, a, b]$ la contrainte c_S qui garantit que (x, b) n'est pas SNS-éliminable par (x, a), si elle existe. La contrainte résiduelle est prioritaire (lignes 4 à 6) ; de cette façon, si elle permet de compenser le surcoût initial unaire, elle nous évite tout travail supplémentaire. Elle est mise à jour aux lignes 9-10. Notons que nous pouvons initialiser le tableau $residues$ avec n'importe quelle contrainte arbitraire et que l'algorithme 1 retourne nécessairement une valeur inférieure ou égale à 0 (ce qui explique l'initialisation de $ocst$ à 1 à la ligne 1).

Nous discutons maintenant de la complexité de PSNS^r tout en faisant l'hypothèse (pour simplifier) que le WCN est binaire. La complexité en espace est $O(nd^2)$ en raison de l'utilisation de la structure $residues$. La complexité en temps de l'algorithme 1 est $O(d)$, et la complexité en temps de l'algorithme 2 est $O(1)$ dans le meilleur des cas (si il est arrêté à la ligne 3) et $O(qd)$ dans le pire des cas, où $q = |\Gamma(x)|$. Sans la ligne 3, la complexité en temps de l'algorithme 3 est $O(nd^2)$ dans le meilleur des cas et $O(nd^3e)$ dans le pire des cas (à noter que $\sum_{x \in vars(P)} |\Gamma(x)|$ est $O(e)$). Bien sûr, l'algorithme 3 peut être appelé à plusieurs reprises à la ligne 6 de l'algorithme 4, ainsi obtient-on une complexité dans le pire des cas en $O(n^2d^4e)$. Cependant, nous avons observé dans nos expérimentations que le nombre d'appels successifs à PSNS^r était très limité en pratique (comme on l'imaginait). En outre, imaginons maintenant que nous appelions AC*-PSNS après l'assignation d'une valeur à une variable (i.e. au cours de la recherche) x. Dans le meilleur des cas (du point de vue de la complexité temporelle), aucune suppression n'est effectué par $W\text{-}AC*$, et on considère donc uniquement le voisinage de x (à la ligne 3 de l'algorithme 3), ce qui donne une complexité en temps en $O(qd^2)$. Ce dernier résultat nous permet d'envisager relativement sereinement l'expérimentation du maintien de AC*-PSNS pendant la recherche.

7 Résultats expérimentaux

Pour démontrer l'intérêt pratique de la suppression des valeurs SNS-éliminables, nous avons mené une expérimentation en utilisant les séries d'instances WCSP disponible à l'adresse `http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/BenchmarkS` et un cluster de Xeon 3.0GHz avec 2 Go de RAM sous Linux. Notre but est d'observer l'efficacité relative de la résolution d'instances WCSP lorsqu'on *maintient* AC*, AC*+PSNS, FDAC, FDAC+PSNS, EDAC, et EDAC+PSNS. Pour l'ordre de sélection des variables au cours de la recherche, nous utilisons l'heuristique statique simple *max degree* qui est indépendante de l'efficacité des algorithmes de filtrage utilisés, comme dans [11] où certaines expériences ont été réalisées avec une forme partielle de SNS appliqué lors d'une étape de pré-traitement.

Series		AC*	AC* +PSNS	FDAC	FDAC +PSNS	EDAC	EDAC +PSNS
celar	#solved	5	4	6	6	6	**7**
#inst=7	cpu	337	231	316	341	344	**461**
	avg-sub	0	3	0	6	0	4
driver	#solved	18	18	19	19	19	**19**
#inst=19	cpu	103	52	39.3	19.7	68.5	**56.8**
	avg-sub	0	1	0	1	0	1
geom	#solved	5	5	5	**5**	5	5
#inst=5	cpu	37.4	12.4	18.8	**11.0**	21.0	12.0
	avg-sub	0	0	0	1	0	0
mprime	#solved	4	8	4	**8**	5	8
#inst=8	cpu	9.82	17.4	11.6	**12.0**	206	21.0
	avg-sub	0	1	0	1	0	1
myciel	#solved	3	3	3	**3**	3	3
#inst=3	cpu	122	77.0	72.9	**34.3**	80.4	37.8
	avg-sub	0	2	0	3	0	3
scens+graphs	#solved	1	3	**8**	7	6	5
#inst=9	cpu	20.4	113	**242**	186	266	19.8
	avg-sub	0	8	0	8	0	8
spot5	#solved	0	0	3	3	3	**3**
#inst=3	cpu			20.5	12.6	20.1	**11.4**
	avg-sub	0	0	0	0	0	0
warehouse	#solved	12	12	18	24	28	**29**
#inst=34	cpu	286	46.2	166	139	53.2	**79.8**
	avg-sub	0	4	0	7	0	27
	#solved	48	53	66	75	75	**79**

TABLE 1 – Résultats obtenus pour différentes séries (une échéance de 1,200 secondes par instance).

Instances		AC*	AC* +PSNS	FDAC	FDAC +PSNS	EDAC	EDAC +PSNS
cap101	cpu	232	42.1	1.6	1.62	1.48	**1.12**
	#nodes	242K	242K	835	835	75	75
	avg-sub	0	1	0	2	0	15
cap111	cpu	>1,200	>1,200	633	162	3.03	**2.74**
	#nodes	–	–	72,924	72,924	439	199
	avg-sub	0	2	0	3	0	12
capmo1	cpu	>1,200	>1,200	>1,200	>1,200	>1,200	**984**
	#nodes	–	–	–	–	–	–
	avg-sub	0	15	0	23	0	64
driverlog02ac	cpu	253	49.5	10.6	**7.99**	19.3	13.5
	#nodes	4,729K	701K	19,454	8,412	19,444	8,402
	avg-sub	0	0	0	0	0	0
driverlogs06	cpu	>1,200	>1,200	187	**61.0**	218	80.1
	#nodes	–	–	2,049K	609K	2,049K	609K
	avg-sub	0	0	0	0	0	0
mprime04ac	cpu	>1,200	30.9	>1,200	**15.7**	>1,200	43.7
	#nodes	–	189K	–	22,373	–	20,381
	avg-sub	0	0	0	1	0	1
myciel5g-3	cpu	38.1	19.6	**3.87**	4.18	4.36	4.57
	#nodes	518K	168K	10,046	9,922	6,159	6,128
	avg-sub	0	2	0	3	0	2
celar7-sub1	cpu	925	820	147	145	135	**86.4**
	#nodes	9,078K	1,443K	732K	91,552	796K	70,896
	avg-sub	0	6	0	9	0	6
graph07	cpu	>1,200	261	**3.11**	3.58	3.86	4.3
	#nodes	6,156K	145K	1,112	647	1,796	1,514
	avg-sub	0	23	0	9	0	4
scen06-24	cpu	>1,200	>1,200	**1,061**	>1,200	>1,200	>1,200
	#nodes	–	–	–	375K	–	–
	avg-sub	0	2	0	6	0	7
spot5-29	cpu	>1,200	>1,200	30.1	**18.0**	47.2	22.5
	#nodes	–	–	343K	174K	352K	185K
	avg-sub	0	0	0	0	0	0

TABLE 2 – Résultats illustratifs obtenus sur certaines instances.

Le tableau 2 montre les moyennes des résultats obtenus sur les différentes séries. Pour chaque série, le nombre d'instances considérées (#inst) est donné au-dessous du nom de la série. Nous avons écarté les instances qui n'ont pas été résolues par au moins un des algorithmes, en moins de 1,200 secondes. Ici, une instance résolue signifie qu'une solution optimale a été trouvée et prouvée être optimale. Le nombre moyen (avg-sub) de valeurs SNS-éliminables supprimées lors de la recherche (à chaque étape) est également indiqué (avec des valeurs arrondies à l'entier le plus proche). Sur les instances RLFAP (CELAR, scens, graphs), l'intérêt d'utiliser PSNS est plutôt chaotique, mais sur les instances (driver, mprime), coloring (myciel, geom), spot et warehouse, on peut constater qu'il y a un avantage clair à l'intégration de PSNS. Dans l'ensemble, le maintien de PSNS est rentable car il offre en général un avantage à la fois en termes d'instance résolues (voir dernière ligne du tableau) et en temps CPU. Le tableau 2 présente les résultats obtenus sur certaines instances représentatives. Il est intéressant de noter que sur les instances warehouse (ici, *cap101*, *cap111* et *capmo1*), l'application de PSNS n'entraîne pas une réduction de la taille de l'arbre de recherche (voir les valeurs de #nodes). Cependant, PSNS permet de réduire la taille des domaines, ce qui rend la propagation des contraintes souples plus rapide. Sur *celar7-sub1*, notons que (maintenir) EDAC+PSNS est seulement environ 50% plus rapide que (maintenir) EDAC alors que le nombre de nœuds a été divisé par 10. Cela signifie que sur de telles instances, PSNS est assez coûteux, ce qui laisse sans doute la place à des optimisations supplémentaires.

8 Conclusion

Dans cet article, nous avons étudié la propriété de substituabilité souple au voisinage pour les réseaux de

contraintes pondérées (WCNs) et analysé les conditions permettant l'identification de valeurs substituables par un algorithme de complexité raisonnable (i.e., polynomial). Nous avons prouvé que, même dans les cas simples, lorsque $k \neq \infty$, le problème de décider si une valeur est souple-substituable à une autre au voisinage est coNP-difficile. Nous avons également étudié les relations entre substituabilité souple au voisinage et cohérence d'arc souple. Enfin, nous avons proposé un algorithme qui exploite des arrêts précoces de boucles, des résidus et un mécanisme d'horodatage, et montré expérimentalement qu'il peut être efficace au cours de la recherche.

Remerciements

Ce travail bénéficie du soutien du CNRS et d'OSEO dans le cadre du projet ISI Pajero.

Références

[1] 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.

[2] B.W. Benson and E.C. Freuder. Interchangeability preprocessing can improve forward checking search. In *Proceedings of ECAI'92*, pages 28–30, 1992.

[3] S. Bistarelli, B. Faltings, and N. Neagu. Interchangeability in soft CSPs. In *Proceedings of CSCLP'02*, pages 31–46, 2002.

[4] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-based CSPs and valued CSPs : Frameworks, properties, and comparison. *Constraints*, 4(3) :199–240, 1999.

[5] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Support inference for generic filtering. In *Proceedings of CP'04*, pages 721–725, 2004.

[6] B.Y. Choueiry, B. Faltings, and R. Weigel. Abstraction by interchangeability in resource allocation. In *Proceedings of IJCAI'95*, pages 1694–1710, 1995.

[7] M. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3) :311–342, 2003.

[8] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, and M. Zytnicki. Virtual arc consistency for weighted CSP. In *Proceedings of AAAI'08*, pages 253–258, 2008.

[9] M. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7-8) :449–478, 2010.

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

[11] S. de Givry. Singleton consistency and dominance for weighted CSP. In *Proceedings of CP'04 Workshop on Preferences and Soft Constraints*, 2004.

[12] S. de Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency : Getting closer to full arc consistency in weighted CSPs. In *Proceedings of IJCAI'05*, pages 84–89, 2005.

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

[14] E.C. Freuder and D. Sabin. Interchangeability supports abstraction and reformulation for multi-dimensional constraint satisfaction. In *Proceedings of AAAI'97*, pages 191–196, 1997.

[15] A. Haselbock. Exploiting interchangeabilities in constraint satisfaction problems. In *Proceedings of IJCAI'93*, pages 282–287, 1993.

[16] S. Karakashian, R. Woodward, B. Choueiry, S. Prestwhich, and E. Freuder. A partial taxonomy of substitutability and interchangeability. Technical Report arXiv :1010.4609, CoRR, 2010.

[17] A.M. Koster. *Frequency assignment : Models and Algorithms*. PhD thesis, University of Maastricht, The Netherlands, 1999.

[18] A. Lal, B.Y. Choueiry, and E.C. Freuder. Neighborhood interchangeability and dynamic bundling for non-binary finite CSPs. In *Proceedings of AAAI'05*, pages 397–404, 2005.

[19] J. Larrosa. Node and arc consistency in weighted CSP. In *Proceedings of AAAI'02*, pages 48–53, 2002.

[20] J. Larrosa and T. Schiex. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, 159(1-2) :1–26, 2004.

[21] S. Martello and P. Toth. *Knapsack Problems : Algorithms and Computer Implementations*. Wiley, 1990.

[22] A. Petcu and B. Faltings. Applying interchangeability techniques to the distributed breakout algorithm. In *Proceedings of CP'03*, pages 925–929, 2003.

[23] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems : Hard and easy problems. In *Proceedings of IJCAI'95*, pages 631–639, 1995.