Extension des cohérences WCSP aux tuples

Djamel E. Dehani Christophe Lecoutre Olivier Roussel

CRIL - CNRS UMR 8188,
Université Lille Nord de France, Artois,
rue de l'universite,
62307 Lens cedex, France
{dehani,lecoutre,roussel}@cril.fr

Résumé

Dans cet article, nous présentons un nouveau type de propriétés pour les réseaux de contraintes pondérées (WCNs pour Weighted Constraint networks). Il s'agit de la cohérence de tuples (TC) dont l'établissement sur un WCN est effectué grâce à une nouvelle opération appelée `TupleProject`. Nous proposons également une version "optimale" de cette propriété, OTC, qui peut être perçue comme une généralisation de OSAC (Optimal Soft Arc Consistency). Le principe sous-jacent à OTC est d'appliquer de manière itérative l'opération `TupleProject` afin de factoriser un coût qui maximise la borne inférieure $w_\emptyset$, sur la base de transferts de coûts entre tuples de différentes contraintes d'arité quelconque.

1 Introduction

Le problème de satisfaction de contraintes pondérées (WCSP) consiste, pour un réseau constitué de variables et de contraintes souples, à trouver une affectation de valeur à chaque variable qui soit optimale, c'est à dire une assignation complète de coût minimal parmi toutes les assignations complètes possibles. Il s'agit d'un problème NP-difficile. Un certain nombre de travaux ont étés menés pour adapter au cadre WCSP des propriétés (et algorithmes efficaces) définies pour le cadre CSP, généralement dans le but de filtrer l'espace de recherche, comme par exemple la cohérence de nœud (NC) ou la cohérence d'arc (AC) [9, 10]. Un certain nombre d'algorithmes de plus en plus sophistiqués ont été proposés au cours des années pour approcher la cohérence d'arc souple idéale : FDAC, EDAC, VAC et OSAC (voir [4]).

Les algorithmes évoqués ci-dessus utilisent des opérations de transfert de coûts, appelées transformations préservant l'équivalence (EPTs). Celles-ci permettent la projection (ou extension) de coûts entre contraintes binaires et contraintes unaires ainsi que la projection de coûts entre contraintes unaires et la contrainte particulière, $w_\emptyset$, représentant un coût à apposer à toute instanciation complète. Nous proposons dans cet article d'étudier une nouvelle opération (EPT), appelée `TupleProject`. Nous l'utilisons pour définir de nouvelles propriétés basées sur le transfert de coûts entre contraintes, y compris lorsque ces contraintes sont d'arité supérieure ou égale à 2. Une nouvelle propriété, la cohérence de tuples (TC), est identifiée, ainsi qu'une version "optimale", OTC, qui peut être perçue comme une généralisation de OSAC (Optimal Soft Arc Consistency).

Pour illustrer l'intérêt du transfert de coûts entre tuples de différentes contraintes (non unaires), considérons un WCN contenant 4 variables x, y, z et t avec pour chaque variable deux valeurs possibles a et b, et 3 contraintes w_{xy}, w_{xyz} et w_{xyt} définies par la figure 1 (pour chaque tuple possible, le coût est donné dans la colonne de droite). Ce WCN est arc-cohérent (AC) et même arc-cohérent souple optimal (OSAC) car il n'existe pas de transformation (classique) permettant d'augmenter la borne $w_\emptyset$ ($w_\emptyset$ fournit un minorant du coût du réseau). En revanche, nous remarquons pour la contrainte w_{xyz} que l'instanciation $\{(x, a), (y, b)\}$ a nécessairement un coût de 1. Par conséquent, ce coût peut être déplacé (sans perte d'équivalence) vers le tuple $\{(x, a), (y, b)\}$ de la contrainte w_{xy} à partir de la contrainte w_{xyz}. De même, pour la contrainte w_{xyt}, nous remarquons que l'instanciation $\{(x, b), (y, a)\}$ a également nécessairement un coût de 1. Par conséquent, ce coût peut être déplacé vers le tuple $\{(x, b), (y, a)\}$ de la contrainte w_{xy} à partir de la contrainte w_{xyz}. On obtient alors le réseau de la figure 2. Nous sommes capables maintenant de transférer un coût de 1 sur les valeurs de x ou y

depuis la contrainte w_{xy}, ensuite, de le transférer vers $w_\emptyset$. Le WCN obtenu (voir figure 3) est équivalent au problème d'origine et a maintenant un minorant $w_\emptyset = 1$ et toutes les contraintes sont uniformément nulles.

x	y	w_{xy}
a	a	1
a	b	0
b	a	0
b	b	1

$w_\emptyset$
0

x	y	z	w_{xyz}
a	a	a	0
a	a	b	0
a	b	a	1
a	b	b	1
b	a	a	0
b	a	b	0
b	b	a	0
b	b	b	0

x	y	t	w_{xyt}
a	a	a	0
a	a	b	0
a	b	a	0
a	b	b	0
b	a	a	1
b	a	b	1
b	b	a	0
b	b	b	0

FIGURE 1 – Réseau initial

x	y	w_{xy}
a	a	1
a	b	1
b	a	1
b	b	1

$w_\emptyset$
0

x	y	z	w_{xyz}
a	a	a	0
a	a	b	0
a	b	a	0
a	b	b	0
b	a	a	0
b	a	b	0
b	b	a	0
b	b	b	0

x	y	t	w_{xyt}
a	a	a	0
a	a	b	0
a	b	a	0
a	b	b	0
b	a	a	0
b	a	b	0
b	b	a	0
b	b	b	0

FIGURE 2 – Réseau obtenu après les premiers transferts

x	y	w_{xy}
a	a	0
a	b	0
b	a	0
b	b	0

$w_\emptyset$
1

x	y	z	w_{xyz}
a	a	a	0
a	a	b	0
a	b	a	0
a	b	b	0
b	a	a	0
b	a	b	0
b	b	a	0
b	b	b	0

x	y	t	w_{xyt}
a	a	a	0
a	a	b	0
a	b	a	0
a	b	b	0
b	a	a	0
b	a	b	0
b	b	a	0
b	b	b	0

FIGURE 3 – Réseau final

2 Préliminaires

Un *réseau de contraintes* (CN) P est défini par un ensemble fini de variables, noté $vars(P)$, et un ensemble fini de contraintes, noté $cons(P)$. Chaque variable x a un domaine associé noté $dom(x)$, qui contient l'ensemble fini des valeurs pouvant être assignées à x ; d représente la taille du plus grand domaine. Chaque contrainte c_S porte sur un ensemble ordonné $S \subseteq vars(P)$ de variables appelé *portée* de c_S. Elle est définie par une relation contenant l'ensemble des tuples autorisés pour les variables de S. L'arité d'une contrainte est le nombre de variables dans sa portée. Une contrainte *unaire* (resp., *binaire*) porte sur 1 (resp., 2) variable(s), et une contrainte *non-binaire* porte sur plus de deux 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 $\forall i \in 1..p, a_i \in dom(x_i)$; chaque a_i est noté $I[x_i]$. Une solution de P est une instanciation complète de P (i.e., l'assignation d'une valeur à chaque variable) qui satisfait toutes les contraintes. Pour plus d'information sur les réseaux de contraintes, voir [7, 11].

Un *réseau de contraintes pondéré* (WCN) W est formé par un ensemble fini de variables $vars(W)$, un ensemble fini de contraintes souples $cons(W)$, et une valeur k qui est soit un entier naturel strictement positif soit $+\infty$. Chaque contrainte *souple* $w_S \in cons(W)$ porte sur un ensemble ordonné S de variables (sa portée) et est définie par une fonction de coût de $l(S)$ vers $\{0, \ldots, k\}$, où $l(S)$ est le produit cartésien des domaines des variables présentes dans S ; pour toute instanciation $I \in l(S)$, on notera le coût de I dans w_S par $w_S(I)$. Pour simplifier, pour toute contrainte (souple) w_S, un couple (x, a) avec $x \in S$ et $a \in dom(x)$ est appelé une *valeur* de w_S. Une instanciation de coût k (noté aussi $\top$) est interdite. Autrement, elle est autorisée avec le coût correspondant (0, noté aussi $\bot$, est complètement satisfaisant). Les coûts sont combinés par l'opérateur binaire $\oplus$ défini par :

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

Dans cet article, nous supposons les réseaux normalisés, c'est à dire qu'il n'existe pas deux contraintes de même portée. L'objectif du problème de satisfaction de contraintes pondéré (WCSP) est, pour un WCN donné, de trouver une instanciation complète de coût minimal. Pour plus d'information sur les contraintes pondérées, voir [1, 12].

Différentes variantes de la cohérence d'arc souple pour le cadre WCSP ont étés proposées durant ces dix dernières années. Il s'agit de la cohérence d'arc souple (AC*) [9, 10], la cohérence d'arc directionnelle complète (FDAC) [2], la cohérence d'arc directionnelle existentielle (EDAC) [6], la cohérence d'arc virtuelle (VAC) [3] et la cohérence d'arc souple optimale (OSAC) [4]. Tous les algorithmes proposés pour atteindre ces différents niveaux de cohérence utilisent des opérations de transfert de coûts (ou transformations préservant l'équivalence) décrites dans la section suivante.

Pour finir, nous appellerons tuple d'une contrainte w_S tout tuple de $l(S)$, et dans un souci de simplicité, nous sup-

poserons donné un WCN W, et nous appellerons simplement contrainte toute contrainte souple de W.

3 EPTs

Certaines opérations permettent le transfert de coûts entre différentes contraintes. Elle sont appelées transformations préservant l'équivalence (EPTs). Nous introduisons d'abord les EPTs classiques, puis nous proposons une opération originale.

3.1 EPTs classiques

Les EPTs classiques sont les opérations `Project` et `UnaryProject` utilisées dans de nombreux algorithmes. Elles permettent de factoriser des coûts entre contraintes de portées différentes tout en garantissant l'équivalence. Pour factoriser les coûts, il faut être capable de les soustraire. Cela se fait à l'aide de l'opérateur $\ominus$ défini comme suit :

$$\forall \alpha, \beta \in \{0, \ldots, k\} \mid \alpha \geq \beta : \alpha \ominus \beta = \begin{cases} \alpha - \beta : \alpha \neq k \\ k : \alpha = k \end{cases}$$

L'algorithme 1 décrit l'opération `Project` qui factorise un coût α depuis les tuples de la contrainte w_S vers la valeur (x, a) de la contrainte w_x.

Algorithm 1: `Project`(w_x, a, w_S, α)

Pré-condition: $x \in S \wedge 0 < \alpha \leq \min_{t \in l(S), t[x]=a} w_S(t)$

1 $w_x(a) \leftarrow w_x(a) \oplus \alpha$
2 **pour chaque** $t \in l(S)$ *t.q.* $t[x] = a$ **faire**
3 $\quad w_S(t) \leftarrow w_S(t) \ominus \alpha$

L'algorithme 2 décrit l'opération `UnaryProject` qui factorise un coût α de la contrainte w_x vers $w_\emptyset$.

Algorithm 2: `UnaryProject`(w_x, α)

Pré-condition: $0 < \alpha \leq \min_{a \in dom(x)} w_x(a)$

1 $w_\emptyset \leftarrow w_\emptyset \oplus \alpha$
2 **pour chaque** $a \in dom(x)$ **faire**
3 $\quad w_x(a) \leftarrow w_x(a) \ominus \alpha$

3.2 Projection entre tuples

Nous proposons une nouvelle opération (EPT) qui généralise les EPT précédentes : la projection entre tuples notée `TupleProject`. L'idée est de pouvoir transférer un coût α entre des tuples de différentes contraintes et d'arité quelconque. La définition de la projection entre tuples est la suivante.

Définition 1 *Soient w_S et $w_{S'}$ deux contraintes telles que $S \subset S'$. L'opération `TupleProject`$(w_S, t, w_{S'}, \alpha)$ consiste à projeter un coût α sur un tuple t de w_S depuis les tuples t' de $w_{S'}$ tels que $t \subset t'$, avec $0 < \alpha \leq \min_{t' \in l(S'), t \subset t'} w_{S'}(t')$. Il s'agit :*

- *d'ajouter α à $w_S(t)$, à l'aide de l'opérateur $\oplus$;*
- *de soustraire α de $w_{S'}(t')$ à l'aide de l'opérateur $\ominus$, et ceci pour chaque $t' \in l(S')$ tel que $t \subset t'$.*

Dans l'exemple de la figure 1, la projection sur le tuple $t = \{(x, a), (y, b)\}$ d'un coût 1 peut se faire grâce à la fonction `TupleProject`$(w_{xy}, t, w_{xyz}, 1)$.

Clairement, `TupleProject` préserve l'équivalence et généralise `Project` et `UnaryProject`. En effet, l'opération `Project` correspond au cas $|S| = 1$ et $|S'| > 1$ tandis que l'opération `UnaryProject` correspond au cas $|S| = 0$ et $|S'| = 1$.

L'EPT `TupleProject` est également décrite par l'algorithme 3 qui factorise un coût α depuis les tuples de $w_{S'}$ vers le tuple t de w_S.

Algorithm 3: `TupleProject`$(w_S, t, w_{S'}, \alpha)$

Pré-condition: $S \subset S' \wedge t \in l(S)$
Pré-condition: $0 < \alpha \leq \min_{t' \in l(S'), t \subset t'} w_{S'}(t')$

1 $w_S(t) \leftarrow w_S(t) \oplus \alpha$
2 **pour chaque** $t' \in l(S') \mid t \subset t'$ **faire**
3 $\quad w_{S'}(t') \leftarrow w_{S'}(t') \ominus \alpha$

La définition 1 présente le transfert de coûts entre deux contraintes dans le cas particulier où le scope de l'une est inclus dans le scope de l'autre mais l'objectif de nos transferts est plus large. En effet, cette définition aboutit très naturellement à des transferts de coûts entre contraintes qui partagent des variables, mais sans relation d'inclusion. De fait, quand deux contraintes w_S et $w_{S'}$ partagent des variables, on peut décomposer les transferts en deux étapes : de S vers $S \cap S'$ et ensuite de $S \cap S'$ vers S' (ces deux transferts obéissant à la définition 1).

4 La cohérence de tuples TC

Avant de définir ce type de cohérence, nous introduisons la notion de *sur-contrainte* d'un ensemble de variables.

Définition 2 *Soit $S \subseteq vars(W)$ un ensemble de variables de W. Une sur-contrainte de S est une contrainte $w_{S'}$ telle que $S \subset S'$. L'ensemble des sur-contraintes de S est noté $\Gamma(S) = \{w_{S'} \in cons(W) \mid S \subset S'\}$.*

Définition 3 *Soient $S \subseteq vars(W)$ un ensemble de variables de W, t un tuple de $l(S)$, et $w_{S'}$ une contrainte de $\Gamma(S)$. L'ensemble $\sigma_{w_{S'}}(t) = \{t' \in l(S') \mid t \subset t'\}$ contient tous les tuples définis sur S' et qui étendent t.*

Définition 4 *Soient $S \subseteq vars(W)$ un ensemble de variables de W, t un tuple de $l(S)$, et $w_{S'}$ une contrainte de $\Gamma(S)$. On dit que $t' \in \sigma_{w_{S'}}(t)$ est un support de t sur $w_{S'}$ ssi $w_{S'}(t') = 0$.*

Quand un tuple n'a pas de support, une factorisation est possible et permet d'en créer un. Soit une contrainte w_S de $cons(W)$ et un tuple t de w_S, un support de t sur une contrainte $w_{S'} \in \Gamma(S)$ est créé en procédant comme suit :

- chercher le tuple $t'' = argmin_{t' \in \sigma_{w_{S'}}(t)} w_{S'}(t')$ qui va devenir le support de t sur $w_{S'}$;
- et exécuter $\texttt{TupleProject}(w_S, t, w_{S'}, w_{S'}(t''))$.

Pour illustrer ce dernier point, nous reprenons notre exemple introductif (voir figure 1). Sur ce WCN, il est clair que le tuple $t = \{(x, a), (y, b)\}$ n'a pas de support sur la contrainte w_{xyz}. Pour créer un support à ce tuple, il suffit d'appliquer l'EPT $\texttt{TupleProject}(w_{xy}, t, w_{xyz}, 1)$.

Définition 5 *Soit un ensemble de variables $S \subseteq vars(W)$. Un tuple $t \in l(S)$ est tuple-cohérent (TC) ssi il possède un support sur chaque contrainte $w_{S'} \in \Gamma(S)$. On dit que S est TC ssi tous les tuples de $l(S)$ sont TC. On dit que W est TC ssi tout ensemble de variables $S \subseteq vars(W)$ est TC.*

Chercher à rendre TC un WCN n'est pas toujours réaliste car, dans certains cas, cela peut nécessiter l'introduction d'une contrainte pour tout sous-ensemble possible de variables du problème, ce qui est exponentiel. C'est la raison pour laquelle nous proposons d'étudier une forme limitée de cette cohérence, notée TC$_r$, en ne cherchant des supports que pour les tuples d'arité inférieure ou égale à une arité limite r donnée. En pratique, r aura une petite valeur, ce qui nous permettra de n'introduire, si nécessaire, que des contraintes d'arité faible.

Définition 6 *Pour tout entier strictement positif r, W est TC$_r$ si $\forall S \subseteq vars(W)$ tel que $|S| \leq r$, S est TC.*

Avec la propriété TC$_r$, chaque fois qu'un tuple t défini sur un ensemble de variables S d'arité $i \leq r$ n'admet pas de support sur une contrainte, une factorisation doit être effectuée vers t. S'il n'existe pas de contrainte w_S dans le WCN W, une nouvelle contrainte doit être ajoutée à W, en initialisant les coûts de tous ses tuples à 0 (avant la projection vers t) : $\forall t \in l(S), w_S(t) = 0$. Il est à noter que pour $r = 1$, TC$_r$ est équivalent à AC^*. En effet, dans ce cas, l'établissement de TC$_r$ consiste à créer pour chaque valeur de chaque variable un support sur les contraintes qui portent sur cette variable.

Établir TC$_r$ peut être réalisé à l'aide d'un premier algorithme appelé $TC1_r$ (voir algorithme 4). Nous commençons d'abord par établir la propriété TC pour les tuples d'arité r, puis les tuples d'arité $r-1$, etc. De fait, nous établissons la propriété TC par ordre décroissant d'arité, car les transferts de coûts s'effectuent de contraintes d'arité j vers des contraints d'arité i avec $j > i$. L'intérêt est qu'un support établi pour un tuple t d'une contrainte d'arité i reste valide lorsque la recherche de supports s'effectue aux niveaux inférieurs à i.

Algorithm 4: TC1$_r$(W : WCN)

```
1 pour i ← r à 0 faire
2     pour chaque S ⊆ vars(W) tel que |S| = i faire
3         pour chaque t ∈ l(S) faire
4             pour chaque w_S' ∈ Γ(S) faire
5                 α ← min_{t'∈l(S'),t⊂t'} w_S(t')
6                 si α > 0 alors
7                     si w_S ∉ cons(P) alors
8                         ajouter la contrainte w_S de
                          coût uniformément nul à W
9                     TupleProject(w_S, t, w_S', α)
```

Proposition 1 *Appliqué à tout WCN W, l'algorithme $TC1_r$ produit un WCN qui est TC$_r$ et qui est équivalent à W.*

Preuve. L'algorithme n'applique que des EPTs, donc le réseau obtenu est équivalent au réseau initial. Les lignes 3 à 9 construisent un support (le tuple qui a le coût α dans $w_{S'}$) pour tout ensemble de variables S tel que $|S| = i$. On peut noter que seuls les coûts des tuples de $l(S)$ peuvent augmenter. Comme la boucle externe est décroissante, une fois un support établi, son coût ne peut plus jamais augmenter et reste donc égal à 0. Autrement dit, une fois établi, un support n'est jamais remis en cause par l'algorithme. La boucle énumérant toutes les arités de r à 0, un support est établi pour tout ensemble S tel que $|S| \leq r$. □

L'algorithme 4 nécessite de considérer pour tout ensemble de variables S toutes les contraintes de $\Gamma(S)$. Une autre stratégie pour établir TC$_r$ consiste à ajouter au préalable toutes les contraintes d'arité inférieure ou égale à r qui n'existeraient pas déjà dans le réseau. Cet ajout systématique permet de simplifier les factorisations en exploitant des propriétés de transitivité et reste envisageable tant que r est faible. Nous obtenons un second algorithme $TC2_r$ (voir algorithme 5).

La figure 4 illustre le principe de ce second algorithme. Soit S_1, S_2 et S_3 trois ensembles de variables tels que $S_1 \subset S_2 \subset S_3$ et $r \geq |S_3| = |S_2| + 1 = |S_1| + 2$. Lorsqu'une factorisation est possible de S_3 vers S_1, elle peut s'effectuer en deux étapes : de S_3 vers S_2 puis de S_2 vers S_1.

Algorithm 5: TC2$_r$(W : WCN)

compléter W avec toutes les contraintes manquantes d'arité inférieure ou égale à r
pour chaque $w_S \in cons(W)$ *tel que* $|S| = r$ **faire**
  pour chaque $t \in l(S)$ **faire**
    pour chaque $w_{S'} \in \Gamma(S)$ **faire**
      $\alpha \leftarrow \min_{t' \in l(S'), t \subset t'} w_S(t')$
      si $\alpha > 0$ **alors**
        `TupleProject`$(w_S, t, w_{S'}, \alpha)$
pour $i \leftarrow r-1$ ***à*** 0 **faire**
  pour chaque $w_S \in cons(P)$ *tel que* $|S| = i$ **faire**
    pour chaque $t \in l(S)$ **faire**
      pour chaque $w_{S'} \in \Gamma(S)$ *t.q.* $|S'| = i+1$ **faire**
        $\alpha \leftarrow \min_{t' \in l(S'), t \subset t'} w_S(t')$
        si $\alpha > 0$ **alors**
          `TupleProject`$(w_S, t, w_{S'}, \alpha)$

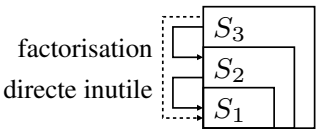

FIGURE 4 – Exploitation de la transitivité des factorisations

Proposition 2 *Appliqué à tout WCN W, l'algorithme TC2$_r$ produit un WCN qui est TC$_r$ et qui est équivalent à W.*

Preuve. La première instruction de l'algorithme assure que toutes les contraintes d'arité inférieure ou égale à r sont présentes dans le réseau. Aussi, quand on peut appliquer `TupleProject` de $w_{S'}$ d'arité a' vers w_S d'arité a avec $a < a' \leq r$, on a alors la garantie qu'il existe une succession de contraintes w_{S_i} pour $i \in a..a'$ telles que $S_{a'} = S'$, $S_a = S$, $\forall i \in a..a'-1, S_i \subset S_{i+1}$ et $|S_i| = i$. De ce fait, on peut remplacer une factorisation directe de $w_{S'}$ vers w_S par une succession de factorisations de $w_{S_{i+1}}$ vers w_{S_i}. □

Avec l'algorithme TC2$_r$, il faut donc d'abord ajouter les contraintes w_S qui portent sur les différents sous-ensembles de variables $S \subseteq vars(P)$ tel que $|S| \leq r$ et $w_S \notin cons(P)$. Pour illustrer ce point, prenons un WCN W comportant seulement deux contraintes ternaires w_{xyz} et w_{wyz}. L'ensemble des contraintes du WCN W' équivalent à W sur lequel nous appliquerons la propriété TC$_r$ avec $r = 2$ est donné par la figure 5.

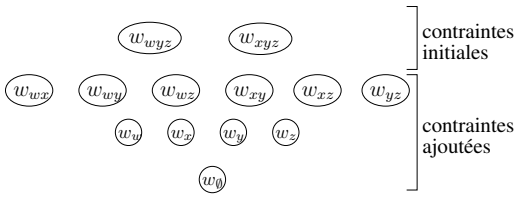

FIGURE 5 – Ajout des contraintes d'arité inférieure ou égale à $r = 2$

En fait, il est possible d'éviter l'introduction de certaines contraintes inutiles. Sur l'exemple précédent, il est inutile d'introduire w_{xw} car elle n'a pas de sur-contrainte. La définition 7 formalise cette remarque.

Définition 7 *Un ensemble de variables $S \subseteq vars(W)$ est dit* isolé *ssi* $\Gamma(S) = \emptyset$.

Clairement, si un ensemble de variables S est isolé alors il est TC. Ainsi, lorsqu'un ensemble de variables S est isolé et que la contrainte w_S est manquante, il n'est pas nécessaire d'ajouter cette contrainte au réseau. Notre réseau de contraintes de la figure 5 peut donc être simplifié en supprimant la contrainte w_{wx}.

Une variante de TC$_r$ consiste à ne pas ajouter de contraintes au réseau, et à simplement créer des supports pour les tuples des contraintes existantes d'arité inférieure ou égale à r. Comme il s'agit d'une version affaiblie de TC$_r$, nous appellerons cette variante TC$_r^w$ (weak TC$_r$). Établir TC$_r^w$ peut s'effectuer sur la base de l'algorithme $TC1_r$ (voir algorithme 4) en s'interdisant simplement l'ajout de contrainte et en ne parcourant que les ensembles S qui sont le scope de contraintes existantes.

5 La cohérence de tuples optimale (OTC)

Nous proposons maintenant une méthode cherchant à tirer le meilleur parti de l'opération `TupleProject`, tout comme OSAC exploite au mieux les EPTs classiques.

Il est facile de montrer que l'application itérée de `TupleProject` n'est pas confluente, ce qui signifie qu'on peut obtenir plusieurs résultats différents selon l'ordre dans lequel on effectue les opérations. D'une part, le WCN obtenu après factorisation n'est pas unique car les coûts peuvent être factorisés de différentes manières, et d'autre part, la borne inférieure $w_\emptyset$ qui est obtenue peut également être différente.

Dans cette section, nous développons la même stratégie que pour OSAC [5], à savoir rechercher un ensemble de projections `TupleProject` qui peuvent être appliquées simultanément de manière à augmenter le plus possible le minorant $w_\emptyset$. Cette recherche est codée sous la forme d'un

problème linéaire. De manière à permettre la résolution de ce problème en temps polynomial, nous autorisons la factorisation de coûts rationnels et pas seulement entier. Nous considérons donc dans ce qui suit que les coûts du WCN sont rationnels.

Comme pour OSAC, on ne cherche pas à déterminer dans quel ordre appliquer les opérations `TupleProject`, ni à vérifier si à chaque étape le WCN obtenu est valide (tous les coûts restent positifs). On considère que les opérations sont effectuées simultanément et notre seule exigence est que le WCN final soit valide et équivalent au réseau initial.

Définition 8 *Étant donné un WCN W, un ensemble d'opérations `TupleProject` est valide si et seulement si, appliquées simultanément, ces opérations transforment W en un WCN W' qui est valide (tous les coûts restent positifs) et équivalent (toute instanciation complète a le même coût dans W et W').*

La cohérence de tuple optimale (OTC) généralise OSAC car elle utilise l'opération `TupleProject` qui subsume les opérations $Project$ et $UnaryProject$ utilisées dans OSAC.

Définition 9 *Un WCN W est tuple-cohérent optimal (OTC) s'il n'existe pas d'ensemble d'opérations `TupleProject` valide qui, appliqué à W, permette d'augmenter le minorant $w_\emptyset$.*

La méthode OTC peut être déclinée de deux manières : soit en n'ajoutant pas de contrainte comme dans OSAC, soit en complétant initialement le réseau avec des contraintes supplémentaires. Dans les deux cas, il est nécessaire en pratique de limiter l'arité des tuples sur lesquels porte $TupleProject$ afin de conserver une complexité raisonnable. Nous nommerons OTC$_r^w$ (weak OTC$_r$) la version qui n'ajoute pas de contrainte, et OTC$_r$ la version qui ajoute des contraintes. Dans les deux cas, r est l'arité maximale considérée par la méthode. Nous présentons ces deux méthodes dans les sections qui suivent.

5.1 Cohérence de tuple optimale restreinte faible OTC$_r^w$

Dans cette version, on s'interdit de modifier l'ensemble de contraintes. Par ailleurs, pour des raisons d'efficacité, on restreindra les opérations $TupleProject$ au cas où au moins l'une des contraintes est d'arité inférieure ou égale à r. Cette version est nommée OTC$_r^w$.

Pour trouver l'ensemble d'opérations `TupleProject` valide qui maximise $w_\emptyset$, on peut définir un problème de programmation linéaire (PL) comme suit. Nous notons par $\alpha_t^{S'}$ le coût transféré depuis une contrainte $w_{S'}$ vers le tuple t de la contrainte w_S avec $t \in l(S)$ par l'opération `TupleProject`$(w_S, t, w_{S'}, \alpha_t^{S'})$. En utilisant la notation $t[S]$ pour représenter la projection du tuple t sur le scope S, le problème linéaire obtenu est donné en figure 6.

La première inéquation traite des contraintes d'arité strictement supérieure à r. Pour chaque tuple de ces contraintes, le coût initial de ce tuple moins les transferts de coûts vers les contraintes d'arité inférieure ou égale à r doit être au final positif ou nul.

La seconde inéquation traite des contraintes d'arité inférieure ou égale à r. Pour chaque tuple de ces contraintes, le coût initial de ce tuple moins les transferts de coûts vers les contraintes d'arité inférieure plus les transferts depuis les contraintes d'arité supérieure doit être au final positif ou nul.

La fonction d'objectif maximise le transfert de coûts vers $w_\emptyset$ qui est la somme des transferts $\alpha_{w_\emptyset}^x$ de la contrainte unaire portant sur x vers $w_\emptyset$.

On notera que dans le système linéaire, les variables α_t^S peuvent être soit positives, soit négatives. Une valeur positive correspond à une projection d'un coût depuis plusieurs tuples vers un seul, tandis qu'une valeur négative correspond à une extension, c'est à dire le transfert d'un coût depuis un tuple unique vers plusieurs tuples. De ce fait, OTC ne se contente pas d'établir des supports mais alterne les opérations de projection et d'extension de manière à obtenir la meilleure borne $w_\emptyset$.

Lorsque les coûts sont rationnels, ce système linéaire se résout en temps polynomial [8].

5.2 Cohérence de tuple optimale restreinte OTC$_r$

Une seconde déclinaison de OTC consiste à ajouter au réseau de départ toutes les contraintes d'arité inférieure ou égale à r et ensuite effectuer les opérations $TupleProject$ en se restreignant au cas ou au moins l'une des contraintes est d'arité inférieure ou égale à r. Cette méthode est nommée OTC$_r$.

Dans ce cas, en utilisant la transitivité des factorisations, le système linéaire peut être simplifié. Ce système est présenté en figure 7.

La première inéquation traite des contraintes d'arité strictement supérieure à r. Pour chaque tuple de ces contraintes, le coût initial de ce tuple moins les transferts de coûts vers les contraintes d'arité égale à r doit être au final positif ou nul.

La deuxième inéquation traite des contraintes d'arité égale à r. Pour chaque tuple de ces contraintes, le coût initial de ce tuple moins les transferts de coûts vers les contraintes d'arité immédiatement inférieure plus les transferts depuis les contraintes d'arité plus grande que r doit être au final positif ou nul.

La troisième inéquation traite des contraintes d'arité inférieure à r. Pour chaque tuple de ces contraintes, le coût initial de ce tuple moins les transferts de coûts vers les

$$\begin{cases} \max : \sum\limits_{x \in vars(P)} \alpha^x_{w_\emptyset} \\ \forall w_S \in cons(W), \text{ t.q. } |S| > r, \forall t \in l(S) : w_S(t) - \sum\limits_{\substack{S' \subset S \\ |S'| \leq r}} \alpha^S_{t[S']} \geq 0 \\ \forall w_S \in cons(W), \text{ t.q. } |S| \leq r, \forall t \in l(S) : w_S(t) - \sum\limits_{S' \subset S} \alpha^S_{t[S']} + \sum\limits_{w_{S'} \in \Gamma(S)} \alpha^{S'}_t \geq 0 \end{cases}$$

FIGURE 6 – Système linéaire généré par OTC^w_r

$$\begin{cases} \max : \sum\limits_{x \in vars(P)} \alpha^x_{w_\emptyset} \\ \forall w_S \in cons(W) \text{ t.q. } |S| > r, \forall t \in l(S) : w_S(t) - \sum\limits_{\substack{S' \subset S, \\ |S'| = r}} \alpha^S_{t[S']} \geq 0 \\ \forall w_S \in cons(W) \text{ t.q. } |S| = r, \forall t \in l(S) : w_S(t) - \sum\limits_{\substack{S' \subset S, \\ |S| = |S'|+1}} \alpha^S_{t[S']} + \sum\limits_{w_{S'} \in \Gamma(S)} \alpha^{S'}_t \geq 0 \\ \forall w_S \in cons(W) \text{ t.q. } |S| < r, \forall t \in l(S) : w_S(t) - \sum\limits_{\substack{S' \subset S, \\ |S| = |S'|+1}} \alpha^S_{t[S']} + \sum\limits_{\substack{S \subset S', \\ |S'| = |S|+1}} \alpha^{S'}_t \geq 0 \end{cases}$$

FIGURE 7 – Système linéaire généré par OTC_r

contraintes d'arité immédiatement inférieure plus les transferts depuis les contraintes d'arité immédiatement supérieure doit être au final positif ou nul.

La fonction d'objectif maximise le transfert de coûts vers $w_\emptyset$ qui est la somme des transferts $\alpha^x_{w_\emptyset}$ de la contrainte unaire portant sur x vers $w_\emptyset$.

Il est clair que OTC_r (avec $r \geq 1$) est également une généralisation de OSAC, et donc OTC_r obtient un minorant $w_\emptyset$ au moins aussi grand que OSAC.

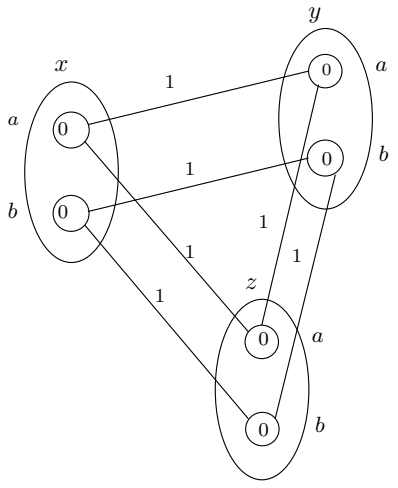

FIGURE 8 – Un WCN OTC_2 cohérent mais non OTC_3 cohérent

L'intérêt de OTC_r est illustré par l'exemple de la figure 8. Ce réseau est OTC_2 et donc OSAC. Par contre, il n'est pas OTC_3 car l'introduction de la contrainte ternaire w_{xyz} va permettre d'augmenter le minorant $w_\emptyset$. En effet, une fois la contrainte w_{xyz} introduite (avec des coûts nuls au départ), on peut transférer le coût de chaque tuple de chaque contrainte binaire vers les tuples correspondant de la contrainte ternaire. Par exemple, le coût 1 assigné au tuple $\{(x,a),(y,a)\}$ est étendu aux deux tuples de $\{(x,a),(y,a),(z,a)\}$ et $\{(x,a),(y,a),(z,b)\}$ de w_{xyz}. À ce moment, les contraintes binaires ont toutes des coûts nuls et les coûts de la contrainte ternaire sont donnés par la figure 9.

x	y	z	w_{xyz}
a	a	a	3
a	a	b	1
a	b	a	1
a	b	b	1
b	a	a	1
b	a	b	1
b	b	a	1
b	b	b	3

FIGURE 9 – La contrainte w_{xyz} après les opérations d'extension

À ce stade, il est direct de factoriser un coût de 1 depuis w_{xyz} vers chacun des tuples de w_{xy} (par exemple), puis de factoriser ce même coût vers w_x (par exemple) pour enfin factoriser 1 vers $w_\emptyset$.

Cet exemple montre que dans certains cas, l'ajout de surcontraintes dans un WCN permet d'augmenter le minorant

$w_\emptyset$.

Il est à noter que le réseau obtenu par OTC$_r$ n'est pas nécessairement TC$_r$, tout comme le réseau obtenu par OSAC n'est pas nécessairement AC.

6 OSAC vs OTC

Dans cette section, nous revenons sur OSAC et OTC, cherchant à établir des rapports entre ces deux cohérences.

Proposition 3 *OSAC est équivalent à OTC$_1^w$.*

Preuve. Le système linéaire généré par OSAC est identique au système linéaire généré par OTC$_1^w$. □

Proposition 4 *Sur l'ensemble des WCNs binaires, OSAC est équivalent à OTC$_r^w$, $\forall r \geq 1$.*

Preuve. Dans le cas des réseaux binaires, que ce soit avec OSAC ou OTC$_r^w$, les seuls transferts possibles sont effectués depuis les contraintes binaires vers les contraintes unaires, puis, depuis les contraintes unaires vers $w_\emptyset$. Aussi, le système linéaire défini par OSAC est le même que celui défini par OTC$_r^w$, sous condition que $r \geq 1$. □

Appliquer OSAC (respectivement, OTC$_r^w$) sur un WCN W permet de calculer un minorant que nous noterons $w_\emptyset^{OSAC}(W)$ (respectivement, $w_\emptyset^{OTC_r^w}(W)$).

Proposition 5 *$\forall r \geq 2$, OTC$_r^w$ est strictement plus fort que OSAC, signifiant que :*

- *pour tout WCN W, $w_\emptyset^{OTC_r^w}(W) \geq w_\emptyset^{OSAC}(W)$*
- *il existe des WCNs W t.q. $w_\emptyset^{OTC_r^w}(W) > w_\emptyset^{OSAC}(W)$*

Preuve. Pour OTC$_r^w$, le système généré contient les inéquations correspondant aux transferts générés par OSAC plus des inéquations pour des transferts potentiels entres tuples. Cela garantit que la valeur du minorant obtenu après l'application de OTC$_r^w$ est supérieure ou égale à celle obtenue après l'application de OSAC. L'exemple introductif à ce papier démontre le caractère strict de cette relation entre OTC$_r^w$ et OSAC. □

Pour finir, nous nous intéressons au cas des WCNs binaires acycliques (i.e., des WCNs dont le graphe de contraintes est acyclique).

Proposition 6 *Soit W un WCN binaire acyclique. Si W est VAC (virtual arc-consistent) alors la valeur de $w_\emptyset$ représente le coût de la solution optimale.*

Preuve. Pour établir VAC, un CN P est construit à partir du WCN W en transformant chaque contrainte souple en une contrainte dure, où seuls les tuples de coût 0 sont autorisés. Comme par hypothèse, W est VAC, cela signifie que W n'est pas détecté incohérent par un algorithme établissant la cohérence d'arc (AC). D'autre part, le graphe de contraintes de W est le même que celui de P, et celui-ci est acyclique. Tout CN acyclique qui est AC est connu pour admettre au moins une solution S. On en déduit que W admet S comme solution de coût 0, auquel il faut ajouter la valeur de $w_\emptyset$. □

Corollaire 1 *Soit W un WCN binaire acyclique. Si W est OSAC alors la valeur de $w_\emptyset$ représente le coût de la solution optimale (puisque un réseau OSAC est nécessairement VAC [3]).*

De ce dernier corollaire, nous pouvons retenir que si le graphe de contraintes est acyclique alors OSAC permet de factoriser le coût optimal vers $w_\emptyset$. Par ailleurs, si on veut établir OTC sur un WCN binaire acyclique, il n'est pas nécessaire d'ajouter de sur-contraintes dans le but d'essayer d'obtenir un minorant plus grand que celui obtenu par OSAC. En revanche, si le WCN n'est pas acylique, l'ajout de sur-contraintes peut éventuellement produire un minorant plus important.

7 Conclusion

Dans cet article, nous avons étudié quelques pistes concernant l'utilisation d'une opération de transfert de coûts généralisant les opérations classiques. Celle-ci permet le transfert de coûts entre deux contraintes toutes deux non unaires. Cela nous a permis de définir une nouvelle cohérence, TC, ainsi que sa version optimisée, OTC. Nous prévoyons d'expérimenter ces nouvelles propriétés dans un avenir proche.

Remerciements

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

Références

[1] 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.

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

[3] 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.

[4] 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.

[5] M.C. Cooper, S. de Givry, and T. Schiex. Optimal soft arc consistency. In *Proceedings of IJCAI'07*, pages 68–73, 2007.

[6] 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.

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

[8] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4) :373–396, 1984.

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

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

[11] C. Lecoutre. *Constraint networks : techniques and algorithms*. ISTE/Wiley, 2009.

[12] P. Meseguer, F. Rossi, and T. Schiex. Soft constraints. In *Handbook of Constraint Programming*, chapter 9, pages 281–328. Elsevier, 2006.