IGC : Une nouvelle consistance partielle pour les CSPs continus

Gilles Chabert, Gilles Trombettoni, Bertrand Neveu

IGC : Une nouvelle consistance partielle pour les CSPs continus

Gilles Chabert Gilles Trombettoni Bertrand Neveu

PROJET COPRIN I3S-INRIA-CERTIS
2004 route des Lucioles
06902 Sophia Antipolis, FRANCE
(chabert|trombe|neveu)@sophia.inria.fr

Résumé

Dans cet article, nous proposons une nouvelle classe de consistances partielles sur les CSPs continus, jusque-là ignorée. Cette approche est motivée par le caractère infructueux des algorithmes de propagation de type AC3 conçus pour des problèmes discrets, et appliqués tels quels en domaine continu, c'est à dire avec la même notion de support (au sens des valeurs réelles). Nous donnons une nouvelle définition, celle de support-intervalle, coïncidant avec une autre abstraction du problème, et déclinons plusieurs propriétés des CSPs liées à cette définition. Nous montrons que seule l'une d'elles (IGC) semble exploitable, et qu'il est possible à partir de l'algorithme de base AC3 de l'obtenir, moyennant le recours à des ROBDD (*reduced ordered binary decision diagrams*), au lieu de simples intervalles, pour la représentation des domaines.

1 Introduction

La programmation par contraintes en domaine continu est une discipline qui s'attache à résoudre des problèmes numériques en utilisant une approche "contraintes". L'une des voies de recherche consiste à transposer des techniques de filtrage de CSPs discrets en des techniques de filtrage de CSPs continus. Cet article s'intéresse à la plus classique d'entre elles, l'arc-cohérence.

Dans cette optique, plusieurs travaux ont déjà été menés, notamment ceux d'Hyvönen [7], Lhomme [8], Faltings [5] et Benhamou [1]. En général ils ont abouti à la conclusion suivante : Si AC3 s'avère efficace, son pendant en domaine continu ne l'est plus du tout. Le fait que la complexité soit polynomiale en le nombre de valeurs du domaine ne peut plus être accepté si cette borne est atteinte en pratique, car la discrétisation implique rapidement un nombre trop important de valeurs. Le moyen classique de pallier cette inadéquation aux valeurs réelles consiste à limiter le filtrage aux bornes des domaines. La propriété obtenue s'appelle la *2B*-cohérence. Nous proposons ici une autre approche.

A partir d'observations sur un exemple, nous introduisons une cohérence partielle plus forte, nommée **IGC**, qui permet de déceler dans la boucle de propagation des blocs de valeurs (sous forme d'intervalles) ne contenant aucune solution. Cette cohérence repose sur une autre vision du problème : un graphe où les intervalles forment les sommets et où les arêtes représentent l'existence de supports. Notre idée est alors d'éliminer au fur et à mesure les intervalles sans support.

Apparaît alors un écueil : la suppression de tels intervalles contient un problème sous-jacent de recherche de clique maximale. Plutôt que de greffer un algorithme générique de graphes pour effectuer cette recherche, nous proposons une méthode (Etiq-AC) adaptée à notre problème, dont l'efficacité tient à son imbrication peu coûteuse dans la boucle de propagation.

Aucune recherche n'est déclenchée explicitement, on se contente de stocker dans des BDDs des informations au cours de la propagation qui permettent de détecter l'absence de clique. La complexité reste exponentielle car nous verrons que le problème est NP-difficile, mais le résultat est très encourageant en pratique.

1.1 Contexte

Notre travail porte sur le problème de satisfaction de contraintes, avec des variables réelles. On note **NCSP** (*numerical constraint satisfaction problem*) une instance de ce problème. Nous limitons dans cet article les contraintes à :

– $y = f(x)$, où f est une fonction réelle monotone.
– $y = x^p$, pour p entier positif quelconque.
– $x = y + z$, $x = y - z$, $x = yz$ et $x = y/z$.

Nous écartons les fonctions trigonométriques par souci de simplicité, mais elles pourraient aussi être prises en compte sans que la théorie change. L'algorithme, par contre, deviendrait beaucoup plus lourd. De plus, pour rendre les exemples plus lisibles, nous utiliserons parfois des contraintes plus élaborées. Le lecteur pourra vérifier que chacun de ces exemples peut être transformé en un système de contraintes primitives sans que cela ne change le propos (il n'y a jamais d'occurrence multiple de variable).

Un NCSP est un triplet (C, V, D) où C est l'ensemble des contraintes, $V = \{x_1, \ldots, x_n\}$ l'ensemble des variables, et $D = D_{x_1} \times \ldots \times D_{x_n}$ le produit cartésien des domaines de resp. $x_1, \ldots, x_n$. La question est de savoir s'il existe un point de D qui satisfait chaque contrainte de C.

Concernant la représentation des domaines, nous serons amenés à manier 2 types de NCPS :
– (C, V, B) est un **NCSP-boîte** si B est le produit de n intervalles fermés de réels.
– (C, V, G) est un **NCSP-gruyère** si G est le produit cartésien de n unions disjointes finies d'intervalles fermés de réels.

1.2 Rappels

Dans les définitions suivantes, on se réfère à un NCSP quelconque (sans supposition sur la nature des domaines).

Définition 1.1 (Projection) *Soit c une contrainte impliquant les variables $x, y_1, ..., y_k$. On appelle projection de c sur x la fonction suivante :*
$\Pi_x^c : D \rightarrow \{v \in D_x \mid \exists (v_1, ..., v_k) \in D_{y_1} \times ... \times D_{y_k},\ c(v, v_1, ..., v_k)$ *est vrai*$\}$

Définition 1.2 (Arc-cohérence)
Soit $P = (C, V, D)$ un NCSP. P est arc-cohérent ssi :
$\forall < c, x > \in C \times V$ *avec x impliquée dans c, $D_x = \Pi_x^c(D)$.*

Définition 1.3 (Partie arc-cohérente) *La partie arc-cohérente d'un NCSP est le sous-NCSP arc-cohérent maximal, au sens de l'inclusion ensembliste (des domaines).*

Remarque 1.1 *On ne sait pas en général quelle est la nature de la partie arc-cohérente d'un NCSP (notamment si pour un NCSP-boîte, elle peut être représentée par un ensemble dénombrable d'intervalles) mais on prouve son existence et son unicité grâce aux deux points suivants :*
- $\emptyset$ est un sous-NCSP arc-cohérent.
- Une union (c.a.d. l'union des domaines pour chaque variable) quelconque de sous-NCSPs arc-cohérents est également un sous-NCSP arc-cohérent.

Nous devons introduire une notion d'approximation pour aboutir à une propriété plus faible que l'arc-cohérence, qu'il soit possible de représenter avec des intervalles, et de calculer avec un algorithme. Nous utiliserons une approximation notée σ-AC (σ désigne une discrétisation des réels), dont la définition est renvoyée en annexe. Cette définition s'applique à des NCSP-gruyères. Les propriétés introduites dans ce papier, comme AC et IGC sont dépendantes de l'approximation sous-jacente et seront renommées σ-AC et σ-IGC lorsqu'on ne pourra pas faire abstraction de σ.

2 Présentation d'IGC

Il a été montré [3] qu'il existe des NCSPs pour lesquels la partie σ-AC possède un nombre d'intervalles de l'ordre du nombre de "flottants" (i.e., $|\sigma|$) contenus dans le domaine initial. La borne de complexité (c.f. proposition A.1 en annexe) est donc atteinte systématiquement pour certains problèmes.

Exemple 2.1 (Exemple-type)
Soit $(\{c_1, c_2\}, \{x, y\}, B)$ le NCSP suivant :
$B = D_x \times D_y = [1, 9] \times [1, 9]$
(c_1) : $(\frac{3}{4}(x - 5))^2 = y$
(c_2) : $x = y$

Dans la figure suivante, les 4 vignettes représentent l'évolution des domaines de x et y au fil des projections.

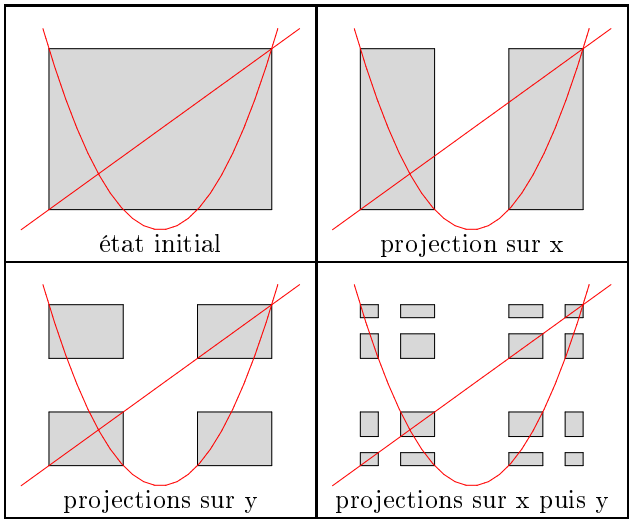


FIG. 1 – Filtrage AC de l'exemple-type (vue "réels")

2.1 Vue macroscopique

L'arc-cohérence appliquée "telle quelle" en domaine continu peut donner lieu à une explosion du nombre

d'intervalles servant à représenter le domaine des variables, et ce problème est lié à la présence de disjonctions. Pour décrire plus précisément le phénomène, considérons par exemple deux variables x et y liées par une contrainte $y = x^2$. Du fait que cette contrainte est disjonctive, un intervalle de y peut se projeter en 2 intervalles pour x, étiquetés par un "+" et un "-" sur la figure 2.

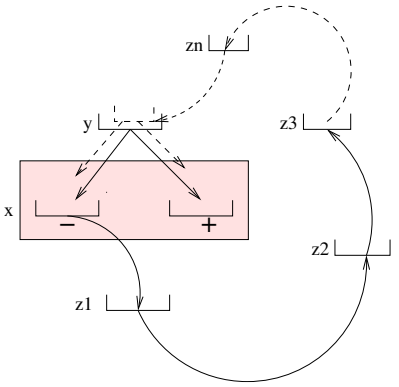


FIG. 2 – Exemple de faux chemin

Considérons maintenant l'intervalle "-" par exemple. Il va se projeter de variable en variable jusqu'à z_n, puis enfin y. Il est possible alors que de nouveau deux intervalles apparaissent pour x, et pourtant, on voit bien que la partie "+" de cette projection ne contient aucune solution puisque l'intervalle projeté "provient" d'une partie "-". C'est ce genre de "faux chemins" qui est à l'origine de l'explosion (chaque projection de y sur x peut doubler le nombre d'intervalles).

Cette observation suggère une vision plus "macroscopique" de notre gruyère. Plutôt que de limiter notre vision aux supports entre valeurs, on peut regarder globalement sur quels intervalles un intervalle possède des supports. On obtient alors un graphe très simplifié, qui permet de détecter qu'un intervalle entier est inconsistant. La définition suivante se place directement dans la cas ternaire (mais les exemples de cette section sont dans le cas binaire) :

Définition 2.1 (Support-intervalle) *Soient x, y et z, trois variables reliées par une contrainte c.*
Soit I (resp. J, K) un intervalle du domaine de x (resp. y, z).
On dit que I, J et K sont compatibles s'il existe $v_i \in I$, $v_j \in J$ et $v_k \in K$ tels que $c(v_i, v_j, v_k)$.

La figure 3 illustre dans l'exemple-type **??**, l'évolution du gruyère du point de vue macroscopique, c'est à dire un graphe où les sommets représentent des intervalles et les arêtes des supports-intervalles. Pour mieux voir cette évolution, nous avons ajouté une variable supplémentaire z, et décomposé c_2 ($x = y$) en deux contraintes, $x = z$ et $z = y$.

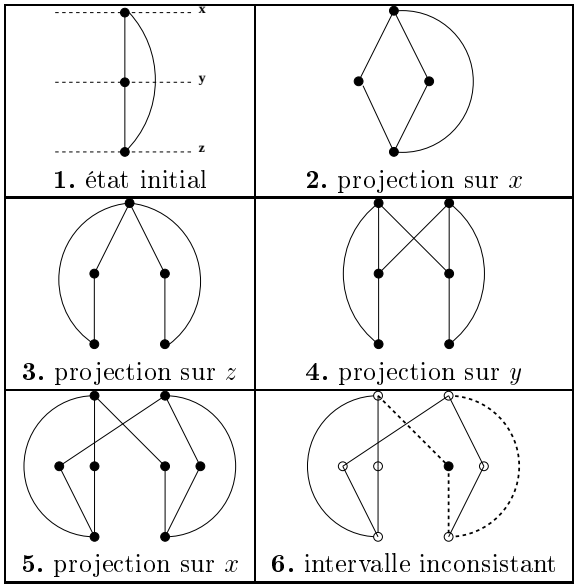


FIG. 3 – Filtrage AC (vue "intervalles")

La figure 3.6 représente le fait que l'intervalle correspondant au point noirci ne peut contenir de solution. Ses supports, apparaissant en pointillés, montrent en effet une incompatibilité.

Notre idée consiste à éliminer les intervalles présentant ce genre d'incompatibilité. On introduit pour cela la notion de *I-clique*.

Définition 2.2 (I-clique) *Soit $P = (C, V, G)$ un NCSP-gruyère.*
Une **I-clique** *de P est un n-uplet constitué d'un intervalle du domaine de chaque variable de P, tel que les n intervalles soient tous compatibles trois par trois (ou deux par deux pour les contraintes binaires).*

Ainsi, dans la figure 3.6 de l'exemple-type, il n'est pas possible de former une I-clique avec l'intervalle noirci[1].

Nous pouvons maintenant définir la propriété que nous cherchons à calculer.

2.2 IGC

Définition 2.3 (IGC) *Un NCSP-gruyère (C, V, G) est* **interval global-cohérent** *(IGC) si tout intervalle du domaine de n'importe quelle variable appartient à une I-clique de G.*

Définition 2.4 (Partie IGC)
Soit P un NCSP-boîte.
On appelle partie IGC de P le NCSP-gruyère formé des intervalles IGC-cohérents de la partie AC de P.

[1] Une I-clique ne correspond pas exactement à une clique maximale dans le graphe qui sert habituellement à représenter un CSP. En effet, il faut ajouter des sommets pour chaque couple de variables, cela étant du à la présence de contraintes ternaires. Un exemple est donné à la figure 7 plus loin.

Remarque sur les I-cliques

La notion de I-clique est plus faible que la notion de "boîte arc-cohérente". Ainsi, même dans un problème arc-cohérent, une I-clique n'est pas forcément une boîte arc-cohérente, et n'en contient pas forcément non plus :

Exemple 2.2 $P = ((c_1, c_2, c_3), \{x_1, x_2, y\}, D_{x_1} \times D_{x_2} \times D_y)$
$D_{x_1} = [-2, 2]$
$D_{x_2} = [-2, 2]$
$D_y = [-2, -1] \cup [1, 2]$
$c_1 : (y - x_1)^2 = [0, 1]$
$c_2 : (y - x_2)^2 = [0, 1]$
$c_3 : x_1 = -x_2$

P est bien arc-cohérent et on peut voir par exemple sur la figure 4 ci-dessous que $([-2, 2], [-2, 2], [1, 2])$ est une I-clique. Cependant, si on "suit" les supports de chaque valeur, on s'aperçoit bien que la boîte $[-2, 2] \times [-2, 2] \times [1, 2]$ n'est pas arc-cohérente. Dans ce cas, elle contient une sous-boîte arc-cohérente, en l'occurrence $[0, 0] \times [0, 0] \times [1, 2]$. Mais il suffit d'ajouter une contrainte, par exemple, $(x_1 - x_2)^2 = 1$ pour qu'il n'y ait plus de sous-boîte arc-cohérente (l'ajout de cette contrainte ne change pas le fait que P soit arc-cohérent, ni les I-cliques).

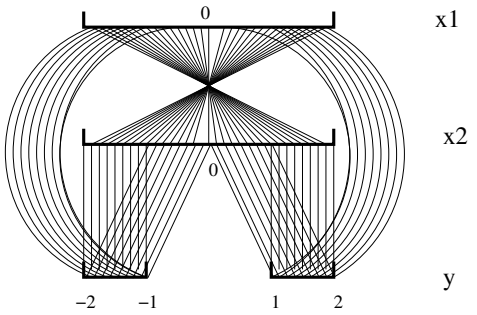


FIG. 4 – Exemple de I-cliques

3 Algorithme

Nous décrivons maintenant notre approche pour calculer la partie IGC d'un NCSP. Notre question devient donc : "Comment isoler un intervalle n'appartenant pas à une I-clique ?".

Nous définissons pour cela un nouvel opérateur de révision, qui en plus de la projection à proprement parler, vérifie les compatibilités (au sens de la définition 2.1) entre les intervalles des domaines.

3.1 Description informelle

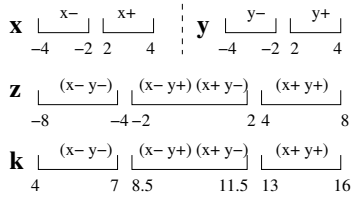
L'algorithme consiste à associer à chaque intervalle du gruyère, une étiquette mémorisant les projections disjonctives dont il est issu depuis le début de la propagation. Quelle que soit la fonction impliquée dans la contrainte (puissance ou division), une projection disjonctive sur x crée forcément deux intervalles de part et d'autre de la valeur 0. Il suffit donc de savoir si un intervalle prend support sur la partie négative x (ce qu'on note x^-) ou sur la partie x^+, pour garder la trace de cette projection.

Lorsqu'on intersecte un intervalle étiqueté x^- avec un intervalle étiqueté x^+, il s'agit donc d'un "faux chemin", et le résultat de cette intersection peut être effacé du domaine.

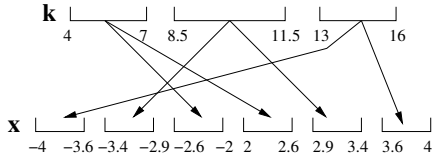
Nous allons dérouler l'algorithme sur un exemple :

Exemple 3.1
$D_x = [-4, 4]$ $D_y = [-4, 4]$ $D_z = [-8, 8]$ $D_k = [4, 16]$
$c_1 : x^2 = k$
$c_2 : y^2 = k$
$c_3 : z = x + y$
$c_4 : k = 0.75 \times z + 10$

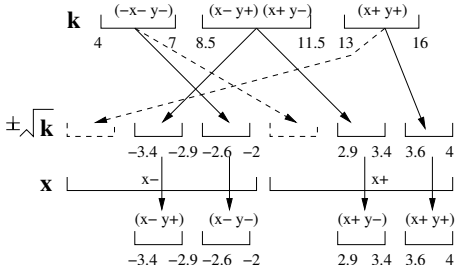
Tout d'abord, la projection de c_1 sur x donnera deux intervalles, $[-4, -2]$ (étiqueté x^-) et $[2, 4]$ (étiqueté x^+). Même observation pour la projection de c_2 sur y. La figure suivante montre le résultat de la projection de c_3 sur z et de c_4 sur k.



Si à ce stade on effectuait une projection classique de c_1 sur x, voici ce que l'on obtiendrait :



La figure suivante montre que la projection étiquetée permet d'éliminer 2 intervalles.



Les 4 intervalles restant contiennent chacun 1 solution du problème, et leur étiquette indique l'unique

boîte monotone (définition plus bas) à laquelle appartient cette solution. Cet exemple montre également qu'il ne suffit pas de mémoriser une information par variable (exemple : x^-), mais bien toutes les combinaisons possibles (comme $\{x^-y^+, x^+y^-\}$), sans quoi la propriété IGC n'est pas obtenu, et surtout, l'explosion observée avec l'arc-cohérence se produit de nouveau.

Nous allons maintenant commencer par expliciter le calcul d'une projection dans le cas d'un gruyère.

3.2 Projections d'unions d'intervalles

Tout d'abord, notons que le calcul des projections se ramène à des évaluations (au sens "intervalles"). Par exemple, la projection de la contrainte $y = exp(x)$ sur y (resp. x) consiste à évaluer $exp(x)$ pour tout $x \in D_x$ (resp. $ln(y)$ pour tout $y \in D_y$) donc à calculer $Exp(D_x)$ (resp. $Ln(D_y)$), où Exp et Ln désignent les extensions aux intervalles des fonctions exp et ln. Dans le cas d'une contrainte non monotone, par exemple $y = x^2$, la projection sur x se ramène à 2 calculs, celui de $\sqrt{y}$ et de $-\sqrt{y}$.

L'étude des projections se limite donc à 4 cas. On notera $D_y = Y_1 \cup \ldots \cup Y_l$ (resp. $D_z = Z_1 \cup \ldots \cup Z_m$) l'union disjointe représentant le domaine de y (resp. z). Dans les 4 cas, le calcul se fait en deux étapes : l'étape *eval*, et l'étape *intersection*.

1. projection sur x de $x = f(y)$
   eval : on calcule $U := F(Y_1) \cup \ldots \cup F(Y_l)$.
   intersection : on calcule $D_x \cap U$.
2. projection sur x de $y = x^p$ (p pair[2])
   eval : on calcule $U := (-\sqrt{Y_1} \cup \ldots \cup -\sqrt{Y_l}) \cup (+\sqrt{Y_1} \cup \ldots \cup +\sqrt{Y_l})$
   intersection : on calcule $D_x \cap U$
3. projection sur x de $x = y \bowtie z$ (avec $\bowtie \in \{+, -, \times, /\}$, et $0 \notin D_z$ si $\bowtie = /$)
   eval : on calcule l'union suivante des (lm) intervalles :
   $U := (Y_1 \bowtie Z_1 \cup \ldots \cup Y_1 \bowtie Z_m) \cup \ldots \cup (Y_l \bowtie Z_1 \cup \ldots \cup Y_l \bowtie Z_m)$
   intersection : on calcule $D_x \cap U$.
4. projection sur x de $x = y/z$, avec $0 \in D_z$
   Elle est basée sur l'arithmétique d'intervalles étendue.Supposons que pour un intervalle de z, noté $[\alpha, \beta]$, on ait $\alpha < 0 < \beta$[3].
   eval : On calcule pour $[\alpha, \beta]$, deux valeurs notées a et b. Si $0 \in D_y$ alors $a = b = 0$. Sinon, a est le minimum des rapports $\{\overline{Y}/\beta$ si $\overline{Y} < 0$, $\underline{Y}/\alpha$ sinon$\}$, lorsque Y décrit $\{Y_1, \ldots, Y_l\}$, et de façon symétrique, b est le minimum des rapports $\{\overline{Y}/\alpha$ si $\overline{Y} > 0$, $\underline{Y}/\beta$ sinon$\}$.
   L'étape "eval" pour les autres intervalles de z (qui ne peuvent contenir 0) se calcule comme dans le cas 3, et produit une union U.
   On calcule enfin $U' := U \cup [-\infty, a] \cup [b, +\infty]$.
   intersection : on calcule $D_x \cap U'$.

Nous allons associer maintenant une étiquette à chaque intervalle traité, mais à titre de "trace", c'est à dire sans impacter les domaines.

3.3 Étiquettes : Définitions et notations

Soient $P = (C, V, B)$ un NCSP, $W = \{w_1, \ldots, w_{n'}\}$ l'ensemble des variables impliquées dans une disjonction de P.
Quitte à réordonner les variables, on peut supposer $V = \{w_1, \ldots, w_{n'}, x_{n'+1}, \ldots, x_n\}$.

On appelle **boîte monotone** un élément de $\Omega = \{[-\infty, 0], [0, +\infty]\}^{n'} \times \{[-\infty, +\infty]\}^{n-n'}$ et on appelle **étiquette** un sous-ensemble de Ω. Une étiquette peut donc prendre $2^{2^{n'}}$ valeurs possibles.

Exemple :
Supposons $V = \{x, y, z\}$ et $W = \{x, y\}$. Un exemple d'étiquette est $\{([-\infty, 0], [0, +\infty], [-\infty, +\infty]), ([0, +\infty], [-\infty, 0], [-\infty, +\infty])\}$, ce qu'on notera de façon plus succincte : $\{(x^-, y^+), (x^+, y^-)\}$. Avec cette dernière notation, on pourra s'autoriser de permuter les variables (et d'écrire donc par exemple $\{(y^+, x^-), (y^-, x^+)\}$). L'important est de pouvoir associer, sans ambiguïté, à toute boîte monotone b contenue dans une étiquette un domaine pour chaque variable. On notera d'ailleurs b_x le domaine associé à x dans b. Ainsi, avec le même exemple, si $b = \{x^-, y^+\}$, alors $b_x = [-\infty, 0]$, $b_y = [0, +\infty]$ et $b_z = [-\infty, +\infty]$.

3.4 Calcul

Si I est un intervalle, on désignera par $\lceil I \rceil$ son étiquette. A chaque nouvelle étape de projection, on calcule les étiquettes du domaine mis à jour grâce au schéma suivant :

Base :
Au départ, chaque intervalle possède pour étiquette Ω lui-même (i.e. l'ensemble des boîtes monotones).

Induction :
Dans la phase "eval", des étiquettes intermédiaires sont définies de la manière suivante (se référer au 3.2 pour les notations) :

1. $\forall i \leq l$, $\lceil F(Y_i) \rceil := \lceil Y_i \rceil$
2. $\forall i \leq l$,
   $\lceil -\sqrt{Y_i} \rceil := \lceil Y_i \rceil \cap \{x^-\} \times \prod_{w \in W\, w \neq x} \{w^-, w^+\}$
   $\lceil +\sqrt{Y_i} \rceil := \lceil Y_i \rceil \cap \{x^+\} \times \prod_{w \in W\, w \neq x} \{w^-, w^+\}$

[2] On supposera $p = 2$, pour alléger les notations.
[3] C'est le cas le plus compliqué, nous ne détaillerons pas ici les autres cas, comme celui où $\alpha = 0 < \beta$.

3. $\forall i \leq l \ \forall j \leq m, \lceil Y_i \bowtie Z_j \rceil = \lceil Y_i \rceil \cap \lceil Z_j \rceil$
4. $[[-\infty, \alpha]] := (\lceil Y_1 \rceil \cup \ldots \cup \lceil Y_l \rceil) \cap \{x^-\} \times \prod_{w \in W w \neq x} \{w^-, w^+\}$
   $[[\beta, +\infty]] := (\lceil Y_1 \rceil \cup \ldots \cup \lceil Y_l \rceil) \cap \{x^+\} \times \prod_{w \in W w \neq x} \{w^-, w^+\}$

Considérons les intervalles de U, mis sous forme d'une union disjointe.
Dans les 4 cas, un intervalle de U peut résulter de la fusion de plusieurs intervalles obtenus par le calcul intermédiaire précédent [4], on le dote alors d'une étiquette égale à l'union des étiquettes de ces intervalles.

Dans la phase intersection, chaque intervalle I obtenu pour la variable sur laquelle on projette résulte de l'intersection d'un intervalle J de U et d'un intervalle $\hat{I}$ du domaine précédent de la variable. On pose alors simplement : $\lceil I \rceil := \lceil J \rceil \cap \lceil \hat{I} \rceil$[5].

3.5 Principe d'Etiq-AC

L'algorithme `Etiq-AC` est exactement `AC3` (ou plutôt son équivalent en domaine continu, donné en annexe), sauf que l'opérateur de révision effectue un travail plus complexe qu'une simple projection : il calcule en plus l'étiquette de chaque intervalle (nous verrons dans la partie implémentation comment le faire de façon efficace) et supprime au fur et à mesure les intervalles ayant des étiquettes vides.

Cette suppression est donc synchronisée avec la projection.

Dans la section 5, on prouve que cet algorithme est correct, i.e, qu'il calcule bien IGC. Cette preuve s'articule autour de 2 propositions :

Correction : La proposition 5.1 énonce qu'un intervalle dont l'étiquette est l'ensemble vide n'appartient à aucune I-clique (et peut donc être éliminé).

Complétude : La proposition 5.3 énonce qu'un intervalle ayant une étiquette autre que l'ensemble vide, appartient bien à une I-clique.

La proposition intermédiaire 5.2 sert pour la preuve de la proposition 5.3, et constitue aussi la clef de voûte de notre implémentation.

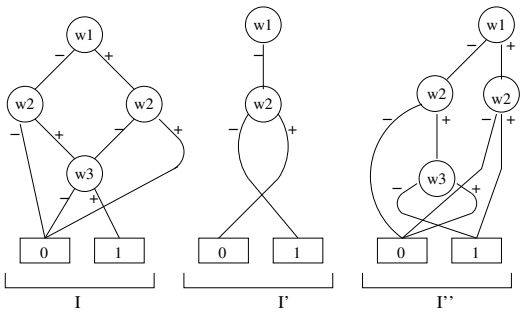
4 Implémentation

4.1 Structure de données

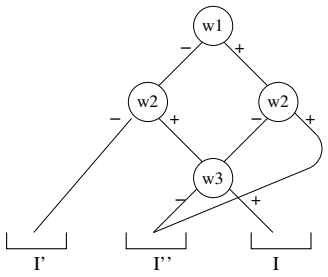
Supposons que notre système comporte 3 variables disjonctives, w_1, w_2 et w_3. Avec nos notations, nous aurons donc $\Omega = \{(w_1^-, w_2^-, w_3^-), (w_1^-, w_2^-, w_3^+), \ldots, (w_1^+, w_2^+, w_3^+)\}$ et rappelons qu'une étiquette est un sous-ensemble de Ω.

Une manière efficace de représenter un sous-ensemble du produit cartésien de n paires est le ROBDD [2].

Mettons par exemple que le domaine d'une variable possède 3 intervalles, I, I' et I'', avec les étiquettes suivantes :
$\lceil I \rceil = \{(w_1^-, w_2^+, w_3^+), (w_1^+, w_2^-, w_3^+)\}$
$\lceil I' \rceil = \{(w_1^-, w_2^-, w_3^-), (w_1^-, w_2^-, w_3^+)\}$
$\lceil I'' \rceil = \{(w_1^-, w_2^+, w_3^-), (w_1^+, w_2^+, w_3^-), (w_1^+, w_2^+, w_3^+)\}$
Avec un ordre fixé entre les variables (par exemple $w_1 < w_2 < w_3$), chaque étiquette peut être représentée par un BDD unique, comme indiqué sur cette figure :



Cette représentation peut être facilement optimisée. Tout d'abord, on peut omettre de stocker pour chaque étiquette les chemins menant à l'état "0", qui s'obtiennent automatiquement par complémentarité. Ensuite, en s'appuyant sur la proposition 5.2, on voit que si un chemin dans le BDD d'un intervalle mène à l'état "1", il mène forcement à l'état "0" dans le BDD des autres intervalles. On peut donc utiliser un seul BDD et stocker aux feuilles l'intervalle correspondant, comme illustré ci-dessous :



4.2 Nouvelle projection (étiquetée)

Un BDD peut se voir comme la représentation d'une fonction f de B^n dans B, avec $B = \{0, 1\}$. Une opération logique (mettons $\vee$) entre 2 BDDs f_1 et f_2 peut être réalisée grâce à une fonction générique `APPLY` [2] qui produit le BDD représentant la fonction $f_1 \vee f_2$.

`APPLY` est basée sur le principe suivant :

– Phase descendante : Parcourir de façon synchronisée le DAG de f_1 et f_2 en générant les combi-

[4] Dans le cas 1, cela ne peut être dû qu'à un problème d'arrondi.

[5] Puisqu'on effectue une projection approximative en élargissant les bornes des intervalles aux valeurs de σ, si deux intervalles deviennent contigus, on doit également les fusionner en calculant l'union des étiquettes.

naisons de valeurs possibles pouvant mener à des feuilles de $f_1 \vee f_2$.
- Calcul des feuilles : Lorsqu'une combinaison notée $w_1, \ldots, w_n$ aboutit à une feuille de $f_1 \vee f_2$, la valeur de cette feuille se calcule en appliquant l'opération logique entre les feuilles correspondantes de f_1 et f_2, i.e. : $(f_1 \vee f_2)(w_1, \ldots, w_n) := f_1(w_1, \ldots, w_n) \vee f_2(w_1, \ldots, w_n)$.
- Phase ascendante : Le DAG est réduit de telle sorte que chaque nœud ait des sous-DAGs différents (*non-redundant test*), et que pour une variable donnée, deux nœuds étiquetés par cette variable n'aient pas à la fois même sous-DAG gauche et même sous-DAG droite (*uniqueness*).

Phase "eval" (cas monotone)

Plaçons-nous d'abord dans le cas **3**, celui de la projection sur x de $x = y \bowtie z$. `APPLY` peut être étendue pour des fonctions de B^n dans n'importe quel ensemble, et ce pour n'importe quel type d'opération. Notre BDD servant à représenter le domaine d'une variable, dans sa version optimisée, est une fonction de B^n dans l'ensemble des intervalles, et on peut appliquer une opération arithmétique entre deux de ces fonctions. Ainsi, le calcul des feuilles devient :

$$(f_1 \bowtie f_2)(w_1, \ldots, w_n) := f_1(w_1, \ldots, w_n) \bowtie f_2(w_1, \ldots, w_n)$$

L'algorithme se déroule de la même manière, avec toutefois la nuance suivante :
Dans la version originale d'`APPLY`, pour réduire le DAG dans la phase ascendante, on est amené à comparer des nœuds et des feuilles. La comparaison entre nœuds utilise une égalité de pointeurs, et la comparaison entre feuilles une égalité de valeurs (0 ou 1). Dans notre version étendue d'`APPLY`, la comparaison entre 2 feuilles J et J' utilise la relation $J \cap J' \neq \emptyset$. Lorsque deux feuilles sont "égales" (lorsqu'elles vérifient la relation), elles sont donc fusionnées, et l'opération de fusion consiste à calculer l'enveloppe $J \cup J'$. Exemple : Dans la figure 5, la combinaison w_1^-, w_2^+, w_3^- donne l'intervalle $[1,2] + [-3,-2]$, soit $[-2,0]$. La combinaison w_1^+, w_2^+, w_3^+ donne l'intervalle $[-2,-1] + [2,3]$, soit $[0,2]$. Ces deux feuilles sont fusionnées en un seul intervalle $[-2,2]$.

Ce n'est donc plus exactement une somme que l'on calcule, mais précisément la phase "eval" de la projection étiquetée décrite au 3.

Le cas **1** de la projection, celui de $x = f(y)$, se déduit facilement de notre discussion. Il correspond à une opération unaire effectuée sur le BDD de x, obtenue comme pour un `not` sur un BDD ordinaire.

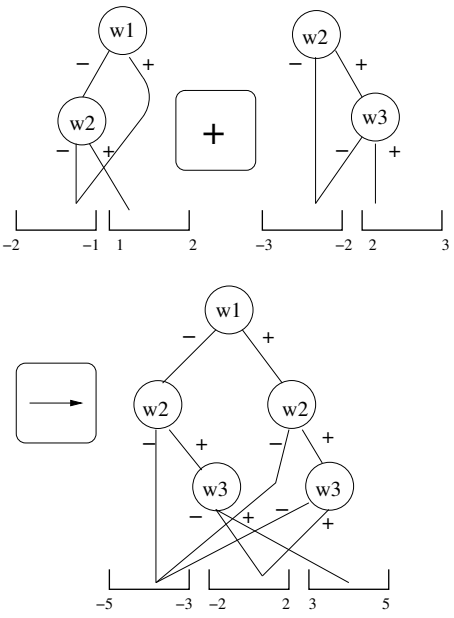


FIG. 5 – Nouvel opérateur Somme

Phase "eval" (cas non-monotone)

Étudions maintenant les cas **2** et **4** de la phase "eval" du calcul de projections. Nous continuons à noter x la variable sur laquelle la contrainte est projetée.

Lors du parcours du DAG de y (et de z dans le cas **4**), l'idée est d'introduire artificiellement dans les combinaisons en entrée la variable x, sauf si celle-ci est déjà présente.

On crée donc une sous-branche x^- et une sous-branche x^+ dans chaque branche du DAG. Dans la sous-branche x^- (resp. x^+), seule la projection $-\sqrt{.}$ (resp. $+\sqrt{.}$) sera autorisée aux feuilles. La figure 6 montre un exemple pour la contrainte $y = x^2$. On suppose dans l'ordre des variables que $w_2 < x$. Le BDD de gauche est celui de y, celui de droite le résultat de la phase "eval". On remarque que x n'apparaît pas dans le sous-DAG w_1^-, w_2^- car les feuilles $[-1,0]$ (pour $-\sqrt{[0,1]}$) et $[0,1]$ (pour $+\sqrt{[0,1]}$) ont été fusionnées.

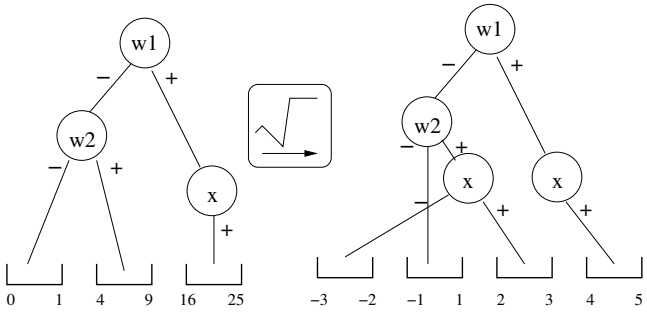


FIG. 6 – Projection de $y = x^2$

On peut vérifier une fois de plus que cela coïncide avec une projection étiquetée.

Phase "intersection"

La phase "intersection" nécessite d'appliquer une opération d'intersection entre le BDD issu de la phase "eval" (mettons $y \bowtie z$) et le BDD de la variable courante, x.

Mais plutôt que de procéder en deux étapes (eval *puis* intersection), on peut encore étendre la fonction `APPLY` à des opérations ternaires. Cette fois, on parcourt de façon synchronisée à la fois le BDD des variables-paramètres et le BDD de x, pour calculer aux feuilles l'opération :

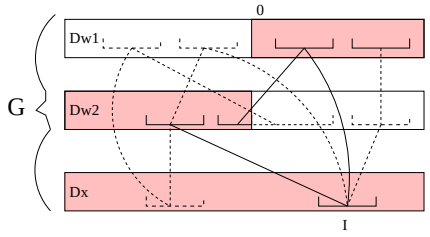
$$(f_x \cap (f_y \bowtie f_z)) :=$$
$$f_x(w_1,\ldots,w_n) \cap f_y(w_1,\ldots,w_n) \bowtie f_z(w_1,\ldots,w_n)$$

On anticipe ainsi sur le fait que les intersections des étiquettes seront calculées au final, en combinant uniquement des intervalles pour lesquels le résultat de cette intersection sera non vide.

Ce triple parcours récursif nous mène donc à effectuer une projection d'intervalles cette fois (et plus d'unions), avec une feuille pour x, y et z. Si l'intervalle obtenu s'avère être vide, le BDD-résultat sera réduit en conséquence à la remontée, de telle sorte qu'il soit bien minimal une fois la projection terminée.
Il remplace ensuite simplement le BDD courant de x.

5 Résultats théoriques

Revenons à un NCSP-gruyère obtenu à une étape quelconque de l'algorithme `AC3`, où les étiquettes on été calculées. Soit x une variable, I un intervalle de son domaine. Nous allons montrer que si une boîte monotone b n'appartient pas à l'étiquette de I, alors le gruyère G' obtenu en remplaçant dans $G \cap b$ le domaine de x par $b_x \cap I$ (on écrira $G' := (G \cap b)_{x \leftarrow I}$) ne contient pas de I-clique. Ceci est illustré sur la figure ci-dessous. On suppose que b est la boîte monotone $\{w_1^+, w_2^-\}$, et que $\lceil I \rceil$ ne contient pas b. La figure montre qu'on ne peut pas former de I-clique avec les arêtes en gras, i.e., celles dont les extrémités sont dans $G \cap b$ (en gris).

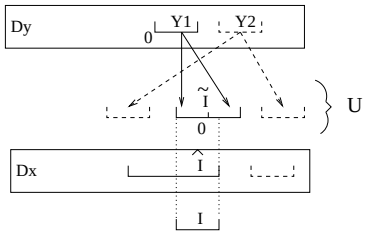


Proposition 5.1 *Soient (C,V,G) un NCSP-gruyère étiqueté, $x \in V$, I un intervalle de D_x, et $b \in \Omega$.*
$b \notin \lceil I \rceil \Longrightarrow (G \cap b)_{x \leftarrow I}$ ne contient pas de I-clique.

preuve :
Base : C'est vrai au départ puisque les étiquettes contiennent toutes les boîtes monotones.
Induction : Il suffit de montrer que l'opération de projection conserve cette propriété. Soit I un intervalle obtenu pour x, la variable sur laquelle on projette. I résulte de l'intersection d'un intervalle $\hat{I}$ de l'ancien domaine de x, et de l'union $\tilde{I}$ de plusieurs intervalles (mettons 2) issus de la phase "eval". Soit $b \in \Omega$ tq $b \notin \lceil I \rceil$.

Si $b \notin \lceil \hat{I} \rceil$, alors la boîte $(G \cap b)_{x \leftarrow \hat{I}}$ ne contient pas de I-clique, et il en est de même pour toute sous-boîte. En particulier, $(G \cap b)_{x \leftarrow I}$ ne contient pas de I-clique.
Si $b \notin \lceil \tilde{I} \rceil$. Par l'absurde, supposons qu'il existe une I-clique dans $(G \cap b)_{x \leftarrow I}$, et observons les 4 cas de projection possibles :

1. $\tilde{I} = F(Y_1) \cup F(Y_2)$. Les seuls intervalle-supports de I sont Y_1 et Y_2 donc la I-clique prend ses valeurs pour y soit dans Y_1, soit dans Y_2, et est donc incluse soit dans $(G \cap b)_{y \leftarrow Y_1}$, soit dans $(G \cap b)_{y \leftarrow Y_2}$. Donc $b \in \lceil Y_1 \rceil$ ou $b \in \lceil Y_2 \rceil$, ce qui implique $b \in \lceil \tilde{I} \rceil$, contradiction.
2. i) Soit Y_1 tel que $-\sqrt{Y_1} \subset \tilde{I}$ et $+\sqrt{Y_1} \subset \tilde{I}$. Si la I-clique prenait ses valeurs dans Y_1 alors $b \in \lceil Y_1 \rceil \Rightarrow b \in \lceil \tilde{I} \rceil$, contradiction (voir figure ci-dessous). ii) Soit Y_2 tel que seul $+\sqrt{Y_2} \subset \tilde{I}$. Si $b_x = [-\infty, 0]$, la I-clique ne peut pas prendre ses valeurs dans Y_2 (puisque $(G \cap b)_{x \leftarrow I} \cap -\sqrt{Y_2} = \emptyset$). Si $b_x = [0, +\infty]$ alors $b \in \lceil Y_2 \rceil \Rightarrow b \in \lceil \tilde{I} \rceil$, contradiction. Même raisonnement dans le cas négatif.
3. Si la I-clique prend ses valeurs dans Y_1 pour y et dans Z_1 pour z, alors $b \in \lceil Y_1 \rceil$ et $b \in \lceil Z_1 \rceil$, donc $b \in \lceil Y_1 \rceil \cap \lceil Z_1 \rceil \Rightarrow b \in \lceil \tilde{I} \rceil$, contradiction.
4. Même idée en combinant les cas 2 et 3. ▲



Proposition 5.2 *Soient (C,V,G) un NCSP-gruyère étiqueté.*
$\forall x \in V$, $\forall I \in D_x$ et $\forall I' \in D_x$:
$I \neq I' \iff \lceil I \rceil \cap \lceil I' \rceil = \emptyset$, i.e., les étiquettes des intervalles d'une variable sont deux à deux disjointes.

preuve : Toujours par induction.
Base : C'est vrai puisque le domaine de chaque variable ne comporte qu'un seul intervalle.

Induction : Considérons 2 intervalles disjoints I et I' dans le nouveau domaine de x. On suppose $I < I'$.
S'il existait auparavant dans le domaine de x deux intervalles disjoints $\hat{I}$ et $\hat{I}'$ tels que $I \subset \hat{I}$ et $I' \subset \hat{I}'$ alors $(\lceil I \rceil \cap \lceil I' \rceil) \subset (\lceil \hat{I} \rceil \cap \lceil \hat{I}' \rceil) = \emptyset$. Sinon, observons de nouveau les 4 cas de projection possibles :

1. I et I' proviennent forcément de la projection d'intervalles différents pour y, formant dans l'étape "eval" deux intervalles disjoints J et J'. Les intervalles de y pris 2 à 2 ont tous au moins une étiquette disjointe, par hypothèse d'induction, donc $(\lceil I \rceil \cap \lceil I' \rceil) \subset (\lceil J \rceil \cap \lceil J' \rceil) = \emptyset$
2. Si $I < 0 < I'$, alors $\lceil I \rceil$ ne contient aucune boîte b telle que $b_x = x^-$ et $\lceil I' \rceil$ ne contient aucune boîte b telle que $b_x = x^+$ donc $\lceil I \rceil \cap \lceil I' \rceil = \emptyset$. Sinon, I et I' proviennent d'intervalles différents de y et on raisonne comme au cas 1.
3. Idem que pour 1. Il suffit d'isoler deux intervalles disjoints pour y ou pour z, et les étiquettes de I et I' hériteront de la propriété voulue.
4. Idem en combinant les cas 2 et 3. ▲

Nous pouvons maintenant montrer facilement que, pour un NCSP arc cohérent, la proposition 5.1 admet une réciproque. Il s'agit toutefois de prendre la précaution suivante : Comme il est possible qu'une projection ne modifie pas le domaine d'une variable, mais modifie son étiquette, lorsque le point fixe de `AC3` est atteint il n'est pas garanti que les étiquettes soient "stables". On prolonge donc artificiellement des projections (sans incidence sur le gruyère) jusqu'à ce que les étiquettes également aient atteint leur point fixe (ce qui arrive forcément après un nombre fini d'itérations, puisqu'une étiquette possède un nombre fini de valeurs et ne fait que décroître).

Exemple 5.1 *Soit* $(C, \{w_1, w_2, x, y\}, \{D_{w_1} \times D_{w_2} \times D_x \times D_y\})$ *un NCSP.* $D_{w_1} = [-2, 2] \quad D_{w_2} = [-2, 2]$
$D_x = [-1, 1] \quad D_y = [-1, 1]$
C contient les 5 contraintes suivantes :
$w_1^2 = [1, 4]; \quad w_2^2 = [1, 4];$
$x = y; \quad x = w_1 - w_2; \quad y = w_1 + w_2$

Si les contraintes sont empilées dans leur ordre de déclaration, les 2 premières réduisent D_{w_1} et D_{w_2} à $[-2, -1] \cup [1, 2]$. Les 3 contraintes suivantes ne modifient pas les domaines. Si le point fixe s'arrête là, IGC n'est pas atteint (il n'y a pas de I-clique sur la figure 7).

Proposition 5.3 *Soient* (C, V, G) *la partie arc cohérente étiquetée (et "stable") d'un NCSP-boîte,* x *une variable et* I *un intervalle de* D_x.
$b \in \lceil I \rceil \iff (G \cap b)_{x \leftarrow I}$ *contient une I-clique.*

preuve : Il reste à montrer $\Longrightarrow$: Si $b \in \lceil I \rceil$, il existe un intervalle compatible avec I dans le domaine de chaque variable (car l'arc cohérence est atteinte) dont l'étiquette contienne b (puisque les étiquettes sont stables), et un seul (d'après la proposition 5.2). On obtient un n-uplet d'intervalles compatibles avec I, mais ce même n-uplet est aussi compatible avec ses autres intervalles, en répétant le même argument. Il s'agit donc bien d'une I-clique, et cette I-clique s'intersecte avec b (ce qu'on montre facilement par induction). ▲

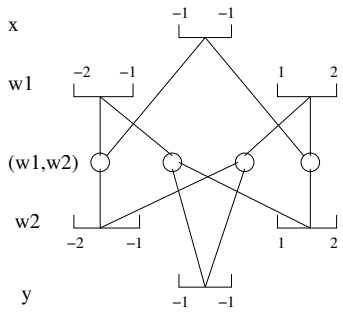


FIG. 7 – Problème du point fixe

De part l'unicité de la partie arc cohérente d'un NCSP, et la convergence vers cette partie de l'algorithme `AC3`, on peut donc associer à toute partie arc cohérente d'un NCSP-boîte, une étiquette à chaque intervalle, et de telle sorte que les étiquettes soient "stables".

Nous pouvons alors énoncer la proposition suivante, qui est une conséquence immédiate des propositions 5.2 et 5.3 :

Proposition 5.4 *Soient* $P = (C, V, B)$ *un NCSP-boîte,* $P' = (C, V, G)$ *la partie arc cohérente de* P.
P' *est IGC* $\iff$ *tout intervalle de* G *possède une étiquette non vide.*

Corrolaire 5.1 *Soit* P *un NCSP-boîte ayant* n' *variables impliquées dans une disjonction.*
Si P' *est la partie IGC de* P*, alors le nombre d'intervalles pour chaque variable de* P' *est borné par* $2^{n'}$.

Très souvent en pratique, la partie σ-IGC est un gruyère nettement plus petit que la partie σ-AC. Il existe des contre-exemples :

Exemple 5.2
Soit le NCSP $((c_1, \ldots, c_m), (x_1, \ldots, x_m), [-2, 2]^m)$
$c_1 : x_1^2 = 1$
$c_2 : (x_2 - x_1)^2 = 1/2$
$c_3 : (x_3 - x_2)^2 = 1/4$
$\ldots$
$c_m : (x_m - x_{m-1})^2 = 1/(2^{m-1})$

Supposons que σ soit une subdivision uniforme de pas $1/2^m$ de $[-2, 2]$. Dans ce problème, on voit alors que la partie σ-IGC se projette en $2^m = 2^{n'}$ intervalles disjoints sur x_m. Ainsi, dans la partie σ-IGC,

le nombre d'intervalles peut atteindre la borne $2^{n'}$ et, d'autre part, être de l'ordre de $|\sigma|$.

Il apparaît, au travers de l'exemple précédent, qu'un algorithme qui calculerait IGC soit a priori exponentiel (en n'). Néanmoins, nous avons pris dans cet exemple σ de telle sorte que $|\sigma| \geq 2^{n'}$. Or, puisqu'on cherche à calculer une consistance à σ-près, on pourrait espérer trouver un algorithme calculant la partie σ-IGC qui reste polynomial en $|\sigma|$, et donc, en utilisant un découpage assez "gros", éviter l'explosion combinatoire qui se produit dans le cas où $2^{n'} \gg \sigma$.

Autrement dit, nous disposerions d'un algorithme qui vérifierait deux critères d'efficacité :

- Théorique : Avoir une complexité en temps polynomiale en $|\sigma|$ (et donc ne jamais présenter de performances moins bonnes que σ-AC3).
- Pratique : Ne produire qu'un petit nombre d'intervalles en calculant la partie σ-IGC.

Hélas, cela n'est pas possible :

Proposition 5.5 *Soit un NCSP-boîte B et σ une subdivision des réels. En supposant que $P \neq NP$, il n'existe pas d'algorithme polynomial pour calculer la partie σ-IGC de B.*

Le théorème repose sur des relations entre le problème 3-SAT [6] et deux problèmes P_1 et P_2 liés aux NCSPs.

P_1 : *Langage :* soit G un NCSP–gruyère et σ une subdivision ; *Question :* est-ce que G est σ–AC *et* contient une I-clique ?

P_2 : *Langage :* soit B un NCSP–boîte et σ un entier positif ; *Question :* la partie σ–*IGC* de G est-elle non vide ?

Le théorème 5.5 découle des deux lemmes suivants :

Lemme 1 *Le problème P_1 est NP-complet.*

Lemme 2 *Le problème P_2 est au moins aussi difficile que le problème P_1.*

Démonstration du théorème 5.5

A partir des lemmes 1 et 2, on déduit immédiatement que le problème P_2 est NP-difficile, d'où le théorème.

Preuve du lemme 1

Nous allons démontrer que le problème 3-SAT peut se réduire polynomialement vers le problème P_1.

Réduction polynomiale. Soit $F = (L, C)$ une instance de 3-SAT ; L est l'ensemble des variables booléennes ; C est l'ensemble des clauses. Soit G' un NCSP issu de la transformation polynomiale, C' l'ensemble des contraintes de G', V' l'ensemble des variables de G', σ la subdivision dans G'.

Pour chaque variable booléenne $x \in L$, on crée une variable $l_x \in V'$ de G' avec un domaine $[-3, 3]$. On choisit la subdivision σ égale à $\{-3, -2, -1, 0, 1, 2, 3\}$.

On crée également une contrainte $l_x^2 = 5$ dans C' de G' (ce qui donne seulement deux intervalles possibles pour l_x : $[-3, -2]$ et $[2, 3]$).

Chaque clause c est une disjonction de 3 littéraux l_1, l_2 et l_3. Elle produit une contrainte $\pm x_{l_1} + \pm x_{l_2} \pm x_{l_3} \geq -5$ dans C' de G'. Le signe est positif (resp. négatif) si le littéral est positif (négatif) dans c.

L'intuition derrière cette réduction est la suivante : si la valuation d'une variable booléenne explique la satisfiabilité d'une clause donnée, on sélectionne l'intervalle correspondant pour la variable correspondante dans la contrainte ternaire associée : $[2, 3]$ si la variable est à *vrai* ; [-3,-2] si elle est à *faux*. Cela assure qu'un des termes de la contrainte ternaire est $[2, 3]$, si bien qu'elle est satisfaite (et σ-AC). En effet, si l'un des trois termes est $[2, 3]$, alors la contrainte est nécessairement satisfaite, indépendamment du domaine des 2 autres variables ([-3,-2] ou [2,3] ou $[-3, -2] \cup [2, 3]$). De plus, calculer la σ-AC de cette contrainte ne réduit (a fortiori supprime) aucun des intervalles des domaines (à cause de l'inégalité qui ne filtre rien).

$P_1 \in NP$.

Vérifier que G' est σ–AC est en $O(m)$ (m est le nombre de contraintes, au plus ternaires). Il suffit en effet d'effectuer toutes les projections (au plus $3m$ projections) possibles et de vérifier pour chacune d'entre elles que l'intervalle de la variable projetée ne peut être réduit. Vérifier qu'un ensemble de n intervalles donnés forment une I-clique se fait en $O(n)$. La réponse à la question de P_1 se teste en temps polynomial, montrant l'appartenance de P_1 à la classe NP.

Il nous reste à démontrer l'équivalence entre (i) une solution S pour n'importe quelle instance F de 3-SAT et (ii) une solution S' pour l'instance G' du problème P_1 obtenue par la réduction présentée ci-dessus.

$(i) \rightarrow (ii)$ Pour construire la solution S' de G', on sélectionne, dans la contrainte ternaire correspondante, l'intervalle (positif ou négatif) selon la valuation des variables booléennes.

L'intuition de la réduction montre directement le résultat. La contrainte $l_x^2 = 5$ de G' restreint le domaine de l_x à $[-3, -2] \cup [2, 3]$. Les contraintes ternaires sont σ-AC car l'intervalle sélectionné produit un terme $[2, 3]$ dans la contrainte et l'inégalité fait que l'on ne peut pas réduire ou supprimer des intervalles : G' est donc σ-AC.

On voit que l'ensemble des intervalles ainsi sélectionnés forment une I-clique : cette I-clique passe par un intervalle de chaque variable et les intervalles sont compatibles entre eux.

$(ii) \rightarrow (i)$ Pour construire la solution S de F, on considère les intervalles sélectionnés dans la I-clique de S'. On effectue l'opération duale de l'autre sens de la démonstration : si l'intervalle est $[-3,-2]$ (resp. $[2,3]$), on value la variable booléenne correspondante à *faux* (resp. *vrai*) dans S.

Comme la I-clique "satisfait" chaque contrainte ternaire, les valuations correspondantes dans F satisfont chacune des clauses, par construction. c.q.f.d.

Preuve du lemme 2

Par contradiction. Si ce n'était pas vrai, alors il existerait un algorithme polynomial pour résoudre P_2. Cet algorithme fournirait également une réponse à P_1 en temps polynomial, ce qui est impossible à cause du lemme 1 (si $P \neq P'$). En effet, s'il existe une partie σ–IGC non vide alors il existe nécessairement un NCSP–gruyère σ–AC contenant au moins une I–clique.

5.1 Autres cohérences d'intervalles

Nous avons à ce stade deux choix possibles :
- Soit calculer σ-AC, avoir une complexité polynomiale (en $|\sigma|$), mais en pratique avoir à gérer un nombre d'intervalles de l'ordre de $|\sigma|$, ce qui est inenvisageable avec $|\sigma|$ très grand.
- Soit calculer σ-IGC, avec un petit nombre d'intervalles en pratique, mais une complexité au pire exponentielle (en n').

Dans cet article, nous avons montré une façon de calculer la partie σ-IGC d'un NCSP. Mais il serait de bon aloi de s'assurer qu'il n'existe pas un filtrage intermédiaire entre σ-AC et σ-IGC qui cumulerait l'avantage des deux critères. Plutôt que de résoudre un problème NP-difficile pour effectuer un filtrage, on resterait alors sur le plan théorique dans un cadre polynomial, tout en évitant sur le plan pratique d'atteindre la "borne du flottant".

Malheureusement, cela semble peu probable. Des contre-exemples peuvent être donnés, lorsqu'on applique un filtrage au niveau "intervalles" assez fort tel que PIC ou Max-RPC [4].

6 Conclusion

A ce jour, un premier prototype d'`Etiq-AC` a été implanté, avec des algorithmes naïfs (notamment pour la fonction `APPLY`). Elle a permis de vérifier ce qui est avancé dans cet article : Le filtrage IGC s'obtient en pratique avec très peu d'intervalles, notamment là où l'arc-cohérence échoue. Les cas pathologiques semblent très artificiels. D'autre part, les temps de calcul sont encourageants, au sens où `Etiq-AC` perd un facteur constant par rapport à la 2B, et nous avons bon espoir de réduire ce facteur en nous appuyant sur une bibliothèque de gestion de BDDs optimisée. Nous disposerons ainsi d'un filtrage par arc-cohérence comme réelle alternative à la 2B.

A σ-AC

Dans cette annexe, nous proposons de définir un équivalent à l'arc-cohérence en domaine continu, avec le souci de rendre cette définition exploitable. Plus précisément, il s'agit de prendre en compte deux contraintes : la représentation des domaines des variables doit recourir à une structure de données qui exclut par exemple la possibilité de manipuler une infinité d'éléments. Ensuite, la propriété doit être calculable, i.e., doit pouvoir être appliquée par un algorithme.

L'idée est d'autoriser autour des points satisfaisant une contrainte, tous les points appartenant à un petit intervalle. On introduit pour cela une subdivision des domaines, notée σ. Cette notion subsume d'ailleurs celle de flottant.

On suppose que la droite réelle achevée (i.e., complétée par $+\infty$ et $-\infty$) est subdivisée par un nombre fini de points[6], $\{f_1, \ldots\}$. Pour tout nombre réel $v \notin \sigma$, on notera $\sigma(v)$ l'unique intervalle $[f_i, f_{i+1}]$ contenant v. Pour $f_i \in \sigma$, $\sigma(f_i)$ peut être choisi comme étant soit $[f_{i-1}, f_i]$, soit $[f_i, f_{i+1}]$, il suffit d'établir une convention. Pour $\sigma(f_i) := [f_{i-1}, f_i]$, on parlera de convention *gauche*, et pour $\sigma(f_i) := [f_i, f_{i+1}]$, de convention *droite*[7].

Définition A.1 (σ-approximation) *Soit $c(x,y)$ une contrainte. On appelle σ-approximation de c la contrainte $c^\sigma(x,y)$ définie par :*
$c^\sigma(x,y) \Longleftrightarrow \exists x_0 \in \sigma(x), y_0 \in \sigma(y)$ *tq* $c(x_0, y_0)$

Définition A.2 (partie σ-arc cohérente) *Soit $P = (\{c_1, \ldots, c_m\}, V, D)$ un NCSP, σ une subdivision des réels. On appelle partie σ-arc cohérente de P la partie arc-cohérente de $(\{c_1^\sigma, \ldots, c_m^\sigma\}, V, D)$*

Il y a donc unicité de la partie σ-arc cohérente. L'intérêt d'avoir introduit σ est dans cette proposition :

Proposition A.1 *Soit P un NCSP, σ une subdivision des réels. La partie σ-arc-cohérente de P est un gruyère où le nombre d'intervalles pour chaque variable est borné par $|\sigma|$.*

preuve : Prenons une variable x, et notons D'_x le domaine de x dans la partie σ-AC de P. Supposons que x soit impliqué dans toutes les contraintes $(c_1, \ldots, c_m)$, et considérons maintenant l'ensemble $D''_x \subset D'_x$ des

[6] Par simplicité, on choisira la même subdivision pour chaque variable, mais ce n'est pas obligé.

[7] Dans les deux cas, $\sigma(-\infty)$ reste le premier intervalle de la subdivision, et $\sigma(+\infty)$ le dernier.

valeurs qui possèdent un support (pour $c_1, \ldots, c_m$) dans le domaine des autres variables de la partie σ-AC de P. Considérons enfin l'ensemble suivant : $U := \bigcup_{v \in D'_x} \sigma(v)$. Clairement, tous les points de U peuvent satisfaire $c_1^\sigma, \ldots, c_m^\sigma$, et ce sont les seuls. Donc $D'_x \subset U$, et par définition $U \subset D'_x$, U est donc un recouvrement de D'_x. Or, on peut extraire de ce recouvrement un sous-recouvrement fini de D'_x (puisque $D'_x \subset D_x$ est borné), où tous les intervalles sont disjoints. En répétant cet argument pour chaque variable, on voit que le gruyère obtenu, est bien σ-arc-cohérent et maximal. ▲

L'algorithme pour calculer la partie σ-arc cohérente d'un NCSP-boîte est donné ci-dessous. Comme il ne manipule que des gruyères, on notera G le produit des domaines. Cet algorithme possède une seule précondition : dans la boîte B (en entrée) chaque borne d'intervalle doit coïncider avec un élément de σ[8].

Procedure 1 σ-`AC3`(NCSP-boîte (C, V, B))

```
    var Q : Queue
 2: var G : Gruyère

 4: G ← B
    for all pairs <c, x> in C × V do
 6:   if x is related by c then
        add <c, x> in Q
 8: while Q is not empty do
      pop a pair <c, x> from Q
10:   G'_x ⟵ σ(Π^c_x(G))
      if G'_x ⊂ G_x then
12:     Propagate((C, V, G), Q, c, x)
        G_x ← G'_x // G_x est le domaine de x dans G
```

A la ligne 10, $\sigma(\Pi_x^c(G))$ signifie $\bigcup_{v \in \Pi_x^c(G)} \sigma(v)$. Les preuve de terminaison et de complétude ne sont pas données ici : on est exactement ramené au cas de figure d'un CSP fini sur lequel on appliquerait `AC3`.

Exemple A.1
Soit le NCSP $(\{c_1, c_2\}, \{x, y\}, [0,4] \times [0,5])$.
$c_1 : y = 2 \times x$
$c_2 : y = x$

La partie AC de ce NCSP est la boîte $[0,0] \times [0,0]$. Si on choisit la subdivision des entiers (donc "à 1 près"), les étapes de projection de σ-`AC3` donneront :

[8] Cette restriction ne pose pas de problème en pratique. Si on souhaite filtrer une boîte à 10^{-6} près il faut simplement élargir au préalable les bornes des domaines au multiple de 10^{-6} le plus proche.

projection	convention gauche	convention droite
$< c_1, x >$	$[0,3] \times [0,5]$	$[0,3] \times [0,5]$
$< c_2, y >$	$[0,3] \times [0,3]$	$[0,3] \times [0,4]$
$< c_1, x >$	$[0,2] \times [0,3]$	fin
$< c_2, y >$	$[0,2] \times [0,2]$	fin
$< c_1, x >$	$[0,1] \times [0,2]$	fin
$< c_2, y >$	$[0,1] \times [0,1]$	fin
$< c_1, x >$	fin	fin

A.1 Complexité

Le compte du nombre de projections se fait comme pour `AC3` (voir [9]). Il y a autant de projections que d'entrées effectuées dans la queue.

La boucle d'initialisation (ligne 4-7) effectue $O(3 \times m) = O(m)$ entrées (les contraintes sont au plus ternaire). Le pire qui puisse se produire est qu'un seul "flottant" (un intervalle élémentaire $[f_i, f_i + 1]$) soit supprimé à chaque projection, chose qui ne peut se produire qu'au plus $|\sigma|$ fois par variable, et que tous les couples liés au couple (contrainte, variable) en cours doivent être réinsérés dans la queue.

Le total est en $O(m|\sigma|)$. Dans le pire des cas, le nombre d'intervalles pour une variable est en $O(|\sigma|)$, ce qui implique que l'opération de projection (ligne 10) est au pire en $O(|\sigma|^2)$, bien que les contraintes soient ternaires.

Au final, on trouve une complexité en $O(m|\sigma|^3)$, similaire à celle en domaines finis.

Références

[1] F. Benhamou, F. Goualard, L. Granvilliers, and J-F. Puget. Revising hull and box consistency. In *ICLP*, pages 230–244, 1999.

[2] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3) :293–318, 1992.

[3] G. Chabert, G. Trombettoni, and B. Neveu. New light on arc consistency over continous domains. Research Report RR-5365, INRIA, 2004.

[4] R. Debruyne and C. Bessière. From restricted path consistency to max-restricted path consistency. In *CP*, pages 312–326, 1997.

[5] B. Faltings. Arc-consistency for continuous variables. *Artificial Intelligence*, 65, 1994.

[6] Michael R. Garey and David S. Johnson. *Computers and Intractability (A guide to the theory of NP-Completeness)*. W.H. Freeman and Company, 1979.

[7] E. Hyvönen. Constraint reasoning based on interval arithmetic—The tolerance propagation approach. *Artificial Intelligence*, 58 :71–112, 1992.

[8] O. Lhomme. Consistency techniques for numeric csps. In *IJCAI*, pages 232–238, 1993.

[9] A. Mackworth and E. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25 :65–74, 1985.