Exploiter les sous-expressions communes dans les CSP numériques

Ignacio Araya, Bertrand Neveu, Gilles Trombettoni

Actes JFPC 2008

Exploiter les sous-expressions communes dans les CSP numériques

Ignacio Araya, Bertrand Neveu et Gilles Trombettoni

INRIA, Université de Nice-Sophia, CERTIS

{Ignacio.Araya, Bertrand.Neveu, Gilles.Trombettoni}@sophia.inria.fr

Résumé

Il est bien connu que la forme des équations est déterminante pour l'efficacité de la résolution de systèmes d'équations et d'inégalités sur les réels par des techniques par intervalles. Peu de transformations automatiques de ces systèmes ont pourtant été proposées jusqu'ici. L'Élimination des Sous-expressions Communes (CSE) est une technique largement utilisée en optimisation de code. En analyse par intervalles, Vu, Schichl, Sam-Haroud et Neumaier exploitent des sous-expressions communes en transformant le système d'(in)équations en un unique graphe sans circuits. Ils estiment que l'impact des sous-expressions communes sur les performances serait uniquement dû à une réduction du nombre des opérations.

Cet article apporte deux contributions principales. Premièrement, en raison de propriétés liées à l'arithmétique par intervalles, nous démontrons théoriquement et expérimentalement que l'exploitation de certaines sous-expressions communes aux équations non seulement réduit le nombre d'opérations, mais aussi apporte des filtrages additionnels lors de la propagation. Deuxièmement, contrairement aux approches existantes, en nous basant sur une meilleure exploitation des opérateurs n-aires d'addition et multiplication, nous proposons un nouvel algorithme I-CSE qui identifie et exploite *toutes* les sous-expressions communes "utiles". Nous montrons sur un échantillon de problèmes tests que I-CSE détecte plus de sous-expressions communes utiles que les approches traditionnelles et conduit généralement à d'importants gains de performance (de parfois plusieurs ordres de grandeur).

1 Introduction

Granvilliers et al. [7] ont étudié plusieurs manières de combiner les méthodes symboliques et par intervalles pour améliorer l'exécution des résolveurs de contraintes par intervalles. Ils ont remarqué que le calcul avec les bases de Gröbner introduit des redondances qui améliorent souvent l'effet de filtrage des techniques par intervalles. L'utilisation de plusieurs formes des équations dans un même système (par exemple les formes naturelles et centrées) a le même effet. Par ailleurs, il est établi que la présence d'occurrences multiples d'une même variable dans une équation donnée réduit la puissance de l'arithmétique par intervalles [14]. C'est pourquoi plusieurs chercheurs appliquent manuellement des transformations symboliques de leur système, comme des factorisations, pour limiter le nombre d'occurrences multiples de variables [12, 7]. L'élimination de sous-expressions communes (CSE) est la principale technique d'optimisation de code [13]. CSE cherche dans le code des sous-expressions communes avec une évaluation identique et les remplace par des variables auxiliaires. Ce remplacement est fait seulement si des gains en temps CPU sont escomptés.

Les outils de calcul formel comme Mathematica [2] ou Maple [8] représentent les équations par des graphes sans circuits (DAG), où les nœuds avec plusieurs parents correspondent à des *sous-expressions communes* (CS). Dans ce cas, CSE est utilisé pour représenter toutes les expressions avec moins de mémoire.

Suivant cette représentation, Schichl et Neumaier ont proposé de représenter dans un seul DAG un système d'(in)équations traitées par des techniques d'analyse par intervalles [15]. Les CS sont les nœuds du DAG avec plusieurs parents et les principaux opérateurs d'analyse par intervalles sont redéfinis sur cette structure de données : évaluation des fonctions, calcul des dérivées, etc. Vu, Schichl et Sam-Haroud détaillent dans [17] comment effectuer la propagation dans le DAG. En particulier, un intervalle est attaché à chaque nœud interne et la propagation est effectuée de manière sophistiquée : deux queues sont gérées, une pour l'évaluation, l'autre pour la contraction (réduction des intervalles ; filtrage), et les opérations de contraction ont la priorité sur les évaluations. Ceux qui ont exploité les CS manuellement ou automatique-

ment [17, 7] pensent que le gain en temps CPU dû aux sous-expressions communes est limité, parce qu'il serait seulement dû à une réduction du nombre des opérations.

La première bonne nouvelle est que l'intérêt de CSE dans la résolution par intervalles est en fait plus grand. Nous établissons clairement dans le présent article quelles CS sont utiles pour réaliser *une meilleure contraction (filtrage)*. C'est le sujet de la section 3. La section 4 présente le nouvel algorithme I-CSE (*Interval CSE*) pour détecter les CS et générer un nouveau système d'(in)équations. Contrairement aux algorithmes existants, et pour une forme donnée de ces équations, I-CSE est capable de trouver toutes les CS *utiles*. Ceci est principalement dû au fait que nous trouvons toutes les CS n-aires maximales correspondant à des sommes et des produits, modulo la commutativité et l'associativité de ces opérateurs, y compris les CS en conflit qui se chevauchent. Enfin, des expérimentations montrent dans la section 6 que, dans tous les systèmes testés, les CS sont extraites en moins d'une seconde. Le système d'équations transformé conduit les stratégies standard de résolution par intervalles qui utilisent `HC4` à d'importants gains de performance (de parfois plusieurs ordres de grandeur).

2 Cadre de l'étude

Les algorithmes présentés dans cet article visent à résoudre des systèmes d'(in)équations ou, plus généralement, des CSP numériques.

Définition 1 *Un CSP numérique (NCSP)* $P = (V, C, B)$ *est composé d'un ensemble* V *de* n *variables et d'un ensemble* C *de contraintes sur ces variables. Chaque variable* $x_i \in V$ *peut prendre une valeur réelle dans l'intervalle* X_i, $\mathbf{B}$ *étant le produit cartésien des domaines (appelé* **boîte**$) X_1 \times ... \times X_n$. *Une solution de* P *est une affectation des variables de* V *qui satisfait toutes les contraintes de* C.

Trouver toutes les solutions d'un NCSP est réalisé par un algorithme de type *branch and prune*. *Branch :* Diviser le domaine d'une variable en deux sous-domaines de manière combinatoire jusqu'à ce qu'une précision fixée soit atteinte. *Prune :* contraction des domaines. Deux types d'algorithmes sont utilisés : des algorithmes d'analyse par intervalles, comme Newton sur intervalles [14], contractent la boîte courante dans toutes les dimensions simultanément et peuvent garantir qu'une boîte contient une solution unique ; des algorithmes de programmation par contraintes sont également utilisés. `HC4` suit une boucle de propagation comme celle de AC3 et traite les contraintes individuellement avec une procédure `HC4-revise` qui réduit les domaines en supprimant les valeurs incompatibles aux bornes des intervalles [1, 9]. Des consistances plus fortes comme 3B [10], similaire à SAC [5] en domaines finis, obtiennent souvent une meilleure performance. À la fin, l'algorithme de résolution trouve une approximation extérieure de toutes les solutions du NCSP.

L'algorithme `HC4-revise` utilise une représentation des contraintes par des arbres, où les feuilles sont des constantes ou des variables, et les nœuds internes correspondent à des *opérateurs de base* : $x + y$, $x \times y$, $sin(x)$, x^c, ... (x et y sont des variables et c est un nombre entier constant). Un intervalle est associé à chaque nœud. `HC4-revise` fonctionne en deux phases. La phase ascendante (appelée ici *évaluation*) est effectuée des feuilles (variables et constantes) vers la racine. Cette phase évalue, de manière récursive, en utilisant l'extension naturelle des fonctions de base, l'intervalle de la sous-expression représentée par le nœud courant de l'arbre (cf. fig. 1-gauche).

La phase descendante (appelée ici *contraction*) parcourt l'arbre de la racine vers les feuilles et applique en chaque nœud un opérateur de contraction (également appelé projection) (cf. fig. 1-droit). L'opérateur de contraction réduit l'intervalle du nœud courant en éliminant les valeurs incompatibles par rapport à l'opérateur de base unaire ou binaire de son nœud père. Sur la fig. 1, les intervalles en gras ont été réduits. Si un intervalle vide est obtenu au cours de la phase de contraction, cela signifie que la contrainte est incompatible avec les domaines initiaux. Les intervalles calculés dans les nœuds internes ne sont pas stockés d'un appel à `HC4-revise` à l'autre, par opposition aux intervalles des feuilles (c'est-à-dire, les variables).

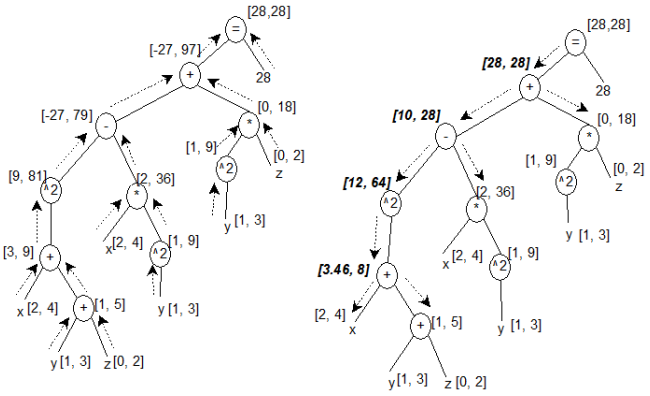

FIG. 1 – Phases d'évaluation et de contraction de l'algorithme `HC4-revise`. L'arbre représente la contrainte : $(x + y + z)^2 - xy^2 + zy^2 = 28$

3 Propriétés de HC4 et de CSE

Nous appelons *Sous-expression Commune* (en abrégé CS) une expression qui apparaît plusieurs fois dans une ou plusieurs contraintes. Cette partie examine quand il est utile d'extraire une CS f d'un NCSP, de créer une nouvelle *variable auxiliaire* v et d'ajouter l'équation $v = f$ dans le système.

Si nous observons attentivement l'algorithme `HC4-revise`, nous pouvons noter que la contraction obtenue par un opérateur de contraction sur une expression donnée f est, en général, partiellement perdue lors de la prochaine évaluation de f. Considérons par exemple sur la fig. 2, une somme $x+y+z$ qui est partagée par deux contraintes A et B (la variable/nœud v_1 représente $x+y+z$ dans A et v_2 représente $x+y+z$ dans B). Sur la fig. 1, la phase de contraction de `HC4-revise` appliquée à la contrainte A contracte v_1 à $[3.46, 8]$. Puis, lorsque la phase d'évaluation de `HC4-revise` est appliquée à la contrainte B, v_2 est fixé à $[3, 9]$. Clairement, l'intervalle de v_2 est plus grand que celui de v_1. Pour éviter cette sur-estimation, l'idée est de remplacer v_1 et v_2 par une variable v et d'ajouter une nouvelle contrainte $v = x+y+z$. Le nouveau système est équivalent à l'original (les deux ont les mêmes solutions) tout en améliorant l'effet de contraction de HC4 en stockant des informations utiles qui peuvent être utilisées dans d'autres contraintes.

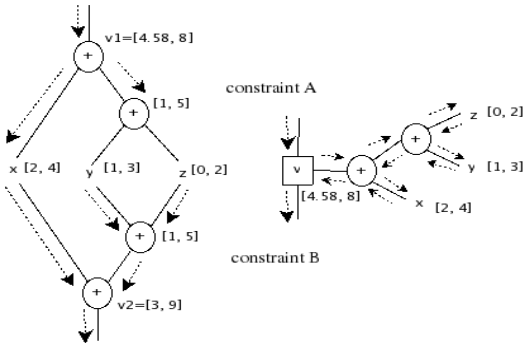

FIG. 2 – Contraction/Evaluation sans et avec CSE.

3.1 Propagation additionnelle

La proposition 1 souligne que `HC4` obtient une meilleure contraction si nous ajoutons, dans un système donné, de nouvelles variables auxiliaires et les équations correspondant aux CS.

Proposition 1 *Soit S un NCSP et S' le NCSP obtenu en remplaçant une CS f, commune entre deux expressions de S, par une variable auxiliaire v, et en ajoutant la nouvelle équation $v = f$ dans S.*
Alors, `HC4` *appliqué à S' produit une boîte contractée B' qui est plus petite ou égale à la boîte B produite par* `HC4` *appliqué à S.*

Preuve. D'abord, on produit un système S_1 en substituant dans S la première occurrence de f par une variable auxiliaire v_1 et la deuxième par une variable auxiliaire v_2. On ajoute les équations $v_1 = f$ et $v_2 = f$. Comme `HC4-revise` travaille sur des graphes sans circuits, `HC4` calcule la 2B-cohérence du système décomposé (c'est-à-dire, le système ternaire équivalent à S où tous les opérateurs sont remplacés par les variables auxiliaires). Il est bien connu que S et S_1 sont équivalents : `HC4` apliqué à S_1 et `HC4` appliqué à S produisent la même boîte contractée B [4]. Finalement, la création de S' revient à ajouter la contrainte $v_1 = v_2$ à S_1. Ainsi, la boîte B' est plus petite ou égale à la boîte B. □

Bien entendu, ce résultat est inutile si la boîte B' est égale à B. Nous ne voulons ainsi remplacer la CS f que si la contraction sur f produit un intervalle qui pourrait être *strictement* plus petit que celui obtenu par la prochaine évaluation de f.

Les *opérateurs de base* qui sont représentés dans une implantation standard de `HC4` sont les suivants : $sin(x)$, $cos(x)$, $tan(x)$, x^c , x, e^x, les opérateurs hyperboliques, $log(x)$, $1/x$, $c+x$, $x+y$, $x-y$, xy, x/y avec c une constante et x, y variables avec pour domaines les intervalles X et Y respectivement. L'analyse présentée ci-dessous met en évidence que l'ensemble suivant des opérateurs non-monotones ou non continus doivent être pris en compte quand ils apparaissent plusieurs fois dans un même système : $sin(x)$, $cos(x)$, $tan(x)$, x^{2c} (c positif et $0 \in X$), $cosh(x)$ avec $0 \in X$, $1/x$ avec $0 \in X$ et les opérateurs binaires $(+,-,\times,/)$.

3.2 Opérateurs unaires

Introduisons d'abord quelques définitions. Une fonction d'*évaluation* sur intervalle associée à une fonction f calcule un intervalle de sortie *conservatif*, c'est-à-dire, l'application de f sur un n-uplet de valeurs pris dans les intervalles d'entrée tombe dans l'intervalle calculé.

Définition 2 *Soit IR l'ensemble de tous les intervalles sur les réels. $F : IR \to IR$, $Y = [\underline{y}, \overline{y}] = f(x)$ est un opérateur d'évaluation associé à un opérateur unaire de base f si : $\forall x \in X, \exists y \in Y$ tel que $f(x) = y$.*

Un opérateur de *contraction* N_F^x associé à une fonction f nous permet de filtrer/contracter, le domaine d'une variable x.

Définition 3 *Soit X le domaine d'une variable x, soit F un opérateur d'évaluation associé à f, et soit Y un intervalle. N_F^x est un* **opérateur de contraction** *de F sur x, si $X' = [\underline{x}, \overline{x}] = N_F^x(Y)$ vérifie :*
$f(\underline{x}) \in Y \wedge f(\overline{x}) \in Y \wedge \forall y \in Y, \forall x \in (X - X') : f(x) \neq y$

Définition 4 *Soit f une fonction définie sur $I(f)$. f est une* **fonction monotone** *sur un intervalle X si :*
$\forall x_1, x_2 \in (X \cap I(f))^2, x_1 \leq x_2 : f(x_1) \leq f(x_2)$ **ou**
$\forall x_1, x_2 \in (X \cap I(f))^2, x_1 \leq x_2 : f(x_1) \geq f(x_2)$

Comme nous l'avons écrit ci-dessus et illustré sur la fig. 2, une condition nécessaire pour remplacer une CS est que la contraction obtenue par un opérateur de contraction sur une expression donnée f soit en partie perdue lors de la prochaine évaluation de f. Plus formellement, la condition 1 pour les opérateurs unaires est la suivante.

$$\exists Y \subseteq F(X), X' = N_F^x(Y) : F(X') \not\subseteq Y \qquad (1)$$

où X est le domaine de la variable x, F est l'opérateur d'évaluation associé à f et N_F^x est l'opérateur de contraction de F sur x.

La proposition suivante donne une condition simple pour identifier une CS inutile pour laquelle aucun filtrage n'est attendu.

Proposition 2 *Soit F l'opérateur d'évaluation associé à un opérateur unaire f. Soit N_F^x l'opérateur de contraction de F sur une variable x de domaine X. Si f est une fonction monotone et continue, on a*

$$\forall Y \subseteq F(X), X' = N_F^x(Y) : F(X') \subseteq Y$$

Preuve. Sans perte de généralité, nous supposons que f est monotone croissante. $X' = N_F(Y)$, alors avec la def. 3 : $f(\underline{x}), f(\overline{x}) \in Y$, où $X' = [\underline{x}, \overline{x}]$. Enfin, avec les defs. 2 et 4, $F(X') = [f(\underline{x}), f(\overline{x})] \subseteq Y$. □

Proposition 3 *Soit F l'opérateur d'évaluation associé à un opérateur unaire f. Soit N_F^x l'opérateur de contraction de F sur une variable x de domaine X. Si f est une fonction non monotone, on a :*

$$\exists Y \subseteq F(X), X' = N_F^x(Y) : F(X') \not\subseteq Y$$

Preuve. La non monotonie de f signifie :

$$\exists x_1, x_2, x_3 \subseteq X^3, x_1 \leq x_2 \leq x_3$$
$$t.q.\ f(x_2) > f(x_1) \wedge f(x_2) > f(x_3)$$

Avec les valeurs x_1, x_2 et x_3 qui satisfont la condition d'existence, on peut supposer que $Y = [f(x_1), f(x_3)]$. Comme $(f(x_2) > f(x_1)) \wedge (f(x_2) > f(x_3))$, $f(x_2) \notin Y$. $X' = N_F(Y)$, donc, avec la def. 1, $[x_1, x_3] \subseteq X'$. Puisque $x_1 \leq x_2 \leq x_3$, $x_2 \in X'$, avec la def. 2, $f(x_2) \in F(X')$. Enfin, $F(X') \not\subseteq Y$. □

Dans notre méthode CSE, seules les CS qui ne vérifient pas la proposition 2 sont susceptibles de produire davantage de filtrage, parce que ces expressions pourraient perdre les informations obtenues par les contractions.

Le domaine d'un nœud interne peut être initialisé avec l'évaluation de ses sous-arbres. À partir du catalogue des opérateurs de base gérés dans `HC4` et mentionné ci-dessus, les CS utiles ne vérifient pas la proposition 2 et vérifient la proposition 3.

Exemple. Soit $X = [-1, 3]$ le domaine d'une variable x et x^2 une expression partagée par deux ou plusieurs contraintes. Supposons que, dans la phase de contraction de `HC4-revise`, le nœud correspondant à une des expressions x^2 soit contracté en : $Y = [3, 4]$. L'application de l'opérateur de contraction sur x donne $X' = [-1, 2]$. Dans l'évaluation suivante de l'expression, $F(X') = [0, 4] \not\subseteq Y$.

3.3 Opérateurs n-aires (sommes, multiplications)

Pour les fonctions de base binaires (n-aires), la condition 1 ci-dessus peut être étendue à la condition suivante 2 :

$$\exists Z \subseteq F(X, Y), X' = N_F^x(Z, Y), Y' = N_F^y(Z, X) : F(X', Y') \not\subseteq Z \quad (2)$$

où X, Y sont les domaines des variables x et y, F est l'opérateur d'évaluation associé à f, N_F^x et N_F^y sont les opérateurs de contraction sur x et y. Cette condition 2 est généralement satisfaite par les opérateurs n-aires $+$ et $\times$ (resp. $-$ et $/$). De nombreux exemples montrent ce résultat (cf. ci-dessous). Cela implique que *les CS d'addition et de multiplication sont classées dans les CS utiles*. Le résultat est dû à une mauvaise propriété "intrinsèque" de l'arithmétique par intervalles. En effet, l'ensemble des intervalles IR n'est pas un groupe pour l'addition. Soit un intervalle $[a, b]$: $[a, b] - [a, b] \neq [0, 0]$ (en fait, $[0, 0] \subset [a, b] - [a, b]$). En outre, $IR - \{0\}$ n'est pas un groupe pour la multiplication, c'est-à-dire, $[a, b]/[a, b] \neq [1, 1]$.

La proposition suivante calcule la taille de l'intervalle que nous pouvons gagner lors du remplacement d'une CS d'addition. Elle estime la largeur Δ qui est perdue par une somme binaire, c'est-à-dire, dans le nœud $+$ correspondant (la variable auxiliaire). Il est intéressant de noter que la borne supérieure de Δ peut être estimée en utilisant seulement les domaines initiaux des variables : $\Delta \leq 2 \times \min(Diam(X), Diam(Y))$.

Proposition 4 *Soit $x + y$ une somme correspondant à un nœud A de l'arbre de représentation de la contrainte. Les domaines de x et y sont les intervalles X et Y respectivement. Supposons que* `HC4-revise` *est exécuté sur la contrainte : dans la phase d'évaluation, A est fixé à l'intervalle $V = X + Y$; dans la phase de contraction, l'intervalle V est contracté en $V_c = [\underline{V} + \alpha, \overline{V} - \beta]$ (avec $\alpha, \beta \geq 0$, la contraction des bornes gauche et droite de V), et X et Y à X_c et Y_c. La différence entre le diamètre de V_c (projection actuelle) et le diamètre de la somme des domaines des variables contractées $X_c + Y_c$ (évaluation suivante) est la suivante :*

$$\Delta = \min(\alpha, Diam(X), Diam(Y), Diam(V) - \alpha) + \min(\beta, Diam(X), Diam(Y), Diam(V) - \beta)$$

Exemple. Soit $X = [0, 1]$ et $Y = [2, 4]$. $V = X + Y = [2, 5]$. Supposons que après avoir appliqué HC4 nous obtenons $V_c = [2 + \alpha, 5 - \beta] = [4, 4]$ ($\alpha = 2$, $\beta = 1$). Avec la proposition 4, nous obtenons $\Delta = 2$. En effet, l'opérateur de contraction donne $X_c = [0, 1]$ et $Y_c = [3, 4]$. Enfin, $X_c + Y_c = [3, 5]$ est $\Delta = 2$ unités plus large que $V_c = [4, 4]$.

Les propriétés liées à la multiplication sont plus difficiles à établir. Des résultats (non présentés ici) ont été obtenus dans les cas où 0 n'appartient pas aux domaines ou lorsque 0 est une borne des domaines.

4 L'algorithme `I-CSE`

L'originalité de notre algorithme `I-CSE` réside dans la façon dont les CS d'addition et de multiplication sont prises en compte.

Tout d'abord, `I-CSE` gère la commutativité et l'associativité de $+$ et $\times$ d'une manière simple grâce à l'*intersection* des expressions. Une **intersection** entre deux sommes (resp. multiplications) f_1 et f_2 produit la somme (resp. la multiplication) de leur termes communs. Par exemple : $+(x, \times(y, +(z, x^2)), 5z) \cap +(x^2, x, 5z) = +(x, 5z)$. Considérons deux expressions $w_1 \times x \times y \times z_1$ et $w_2 \times y \times x \times z_2$ qui partagent $x \times y$. Nous sommes capables de voir ces deux expressions comme $w_1 \times (x \times y) \times z_1$ et $w_2 \times (x \times y) \times z_2$ car $\times(w_1, x, y, z_1) \cap \times(w_2, y, x, z_2) = \times(x, y)$.

Deuxièmement, contrairement aux algorithmes de CSE existants, `I-CSE` gère des sous-expressions *conflictuelles*. Deux CS f_a et f_b incluses dans f sont **en conflit** (ou **conflictuelles**) si $f_a \cap f_b \neq \emptyset$, $f_a \not\subseteq f_b$ et $f_b \not\subseteq f_a$. Un exemple de CS conflictuelles apparaît dans l'expression $f : x \times y \times z$, contenant les CS $x \times y$ et $y \times z$. Comme $x \times y$ et $y \times z$ ont une intersection non vide y, il n'est pas possible de les remplacer toutes deux dans f.

Nous utilisons une représentation du système par un DAG obtenu par la fusion des nœuds identiques des arbres n-aires associés aux équations originales. Les racines de ce DAG correspondent aux équations initiales, les feuilles correspondent à des variables et des constantes, chaque nœud interne f correspond à un *opérateur* ($+, \times, sin, exp$, etc) appliqué à ses *fils* $t_1, t_2..., t_n$. f représente aussi l'expression $f(t_1, t_2..., t_n)$ et $t_1, t_2..., t_n$ les *termes* de l'expression.

Les opérateurs $+$ et $\times$ sont considérés comme opérateurs **n-aires**. Ils comprennent $-$ et $/$. Par exemple, l'expression n-aire $x^2y/(2-x)$ est considérée comme $\times(x^2, y, 1/(+(2, -x)))$.

Nous illustrons `I-CSE` avec le système de deux équations suivant :

$$\begin{aligned} x^2 + y + (y + x^2 + y^3 - 1)^3 + x^3 &= 2 \\ \frac{(x^2 + y^3)(x^2 + cos(y)) + 14}{x^2 + cos(y)} &= 8 \end{aligned}$$

4.1 Étape 1 : Génération du DAG

Au départ, chaque équation du système est représentée par un arbre n-aire, où les nœuds représentent les opérateurs et les feuilles représentent les variables et les constantes.

Deux nœuds sont fusionnés s'ils correspondent à une expression partagée par plusieurs parents, par exemple le nœud 11 dans la fig. 3. L'algorithme que nous utilisons est présenté, par exemple, dans Flajolet et al. [6]. C'est essentiellement une procédure ascendante qui identifie l'égalité des sous-arbres avec des identifiants uniques pour les nœuds. Deux nœuds sont équivalents ssi ils ont le même identifiant.

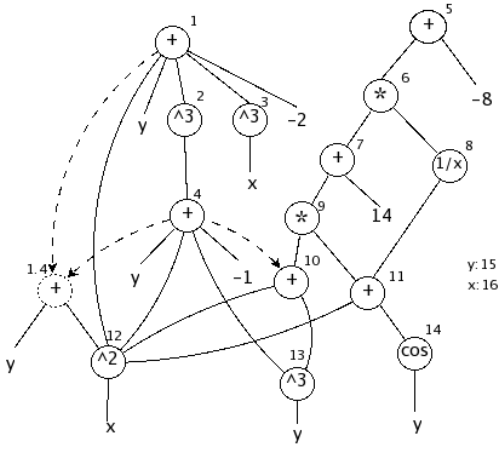

FIG. 3 – Le DAG obtenu après les 2 premières étapes de `I-CSE`. Étape 1 : le système est transformé en un DAG (correspondant à l'ensemble de tous les nœuds à l'exception de 1.4, ainsi que les arcs en traits pleins). Tous les sous-arbres apparaissent une seule fois dans cette représentation (pour rendre la figure plus lisible, nous n'avons pas représenté la fusion des occurrences multiples des variables ; notamment, y serait le nœud 15 et x le nœud 16). Étape 2 : les nœuds $+$ (resp. les nœuds $\times$) sont intersectés deux à deux, d'où la création du nœud 1.4 et des trois *arcs d'inclusion* en pointillés.

4.2 Étape 2 : Intersection deux à deux des sommes et des multiplications

L'étape 2 intersecte deux à deux les nœuds correspondant aux sommes n-aires (avec 2 termes ou plus) d'une part, et aux multiplications n-aires d'autre part. Cette étape crée des *nœuds d'intersection* qui correspondent à des CS. Les nœuds d'intersection sont liés par des arcs spéciaux, appelé *arcs d'inclusion*, à leurs parents.

Si une expression d'intersection n'est pas déjà présente dans le DAG, un nœud correspondant $f_\cap$ est créé. Il représente l'intersection des deux nœuds f_1 et f_2. Sinon, des arcs d'inclusion sont simplement ajoutés de f_1 et f_2 vers le nœud existant.

Par exemple, sur la fig. 3, le nœud 1.4 est obtenu par l'intersection des nœuds 1 et 4, et on crée les arcs d'inclusion des nœuds 1 et 4 vers le nœud 1.4. Cela signifie que 2 (c'est-à-dire, y et x^2) parmi les termes de la somme/nœud 4 sont communs avec 2 parmi les termes de la somme/nœud 1. (Il est à noter que les deux termes sont à des places différentes dans les nœuds d'intersection.) Le nœud 10 correspond à l'intersection entre les nœuds 4 et 10 (en fait le nœud 10 est inclus dans le nœud 4), mais il a déjà été créé à la première étape.

L'étape 2 est essentielle car elle fait apparaître des CS modulo la commutativité et l'associativité des opérateurs $+$ et $\times$, mais aussi parce qu'elle permet à `I-CSE` de créer au plus un nombre quadratique de CS.

Les nœuds d'intersection et les arcs d'inclusion sont

d'une importance cruciale pour le traitement des CS conflictuelles. Ils permettent d'obtenir toutes les CS (en stockant les expressions maximales des intersections), avant de les ajouter dans le DAG[1].

4.3 Étape 3 : Intégration des nœuds d'intersection dans le DAG

À cette étape, tous les nœuds d'intersection sont intégrés dans le DAG pour créer le DAG définitif.

La procédure est descendante en suivant les arcs d'inclusion. Chaque nœud f est traité pour intégrer, dans le DAG final, les CS atteintes par des arcs d'inclusion.

S'il n'existe pas de CS incluses dans f qui partagent de termes, chaque CS est remplacée dans f et les associations respectives sont modifiées. Par exemple, sur la fig. 3, le nœud 1.4 était lié au nœud 1 par un arc d'inclusion. Sur la fig. 5, l'étape 3 transforme l'arc d'inclusion en un arc simple. De plus, pour maintenir l'équivalence entre le DAG et le système d'équations, on supprime les arcs du nœud 1 vers les fils du nœud 1.4.

Sinon, il existe des CS conflictuelles dans l'expression f (c'est-à-dire il existe des CS f_a et f_b, incluses dans f, telles que $f_a \cap f_b \neq \emptyset$, $f_a \not\subseteq f_b$ et $f_b \not\subseteq f_a$). Dans ce cas, un ou plusieurs nœuds équivalents à f sont créés $(f_1, f_2, ..., f_r)$ tels que : $f, f_1, f_2, ..., f_r$ comprennent toutes les CS incluses dans f. Par exemple, le nœud 4 associé à l'expression $x + y^2 + y^3 + 1$, sur la fig. 3, est un nœud avec deux CS en conflit. L'étape 3 crée deux nœuds redondants (4 et 4b sur la fig. 5).

Un algorithme glouton a été conçu pour générer un petit nombre r de nœuds équivalents, avec un petit nombre de fils. r est toujours inférieur au nombre de CS incluses dans f. Nous illustrons cet algorithme glouton qui gère les CS conflictuelles sur un exemple plus compliqué dans lequel une expression (nœud) $u = s+t+x+y+z$ contient 3 CS en conflit : $v_1 = s+t$, $v_2 = t + x$, $v_3 = y + z$ (fig. 4a). L'algorithme glouton fonctionne en deux étapes. Dans la première étape, plusieurs occurrences de u sont générées jusqu'à ce que toutes les CS soient remplacées. Sur l'exemple, ont crée $u = v_1 + x + v_3$ et $u_1 = s + v_2 + y + z$. La deuxième étape traite toutes les équations redondantes créées à l'étape précédente. Elle essaie d'introduire d'une manière gloutonne des CS dans chaque équation pour obtenir une équation plus courte qui améliore le filtrage. Sur l'exemple, elle transforme $u_1 = s + v_2 + y + z$ en $u_1 = s + v_2 + v_3$. Les résultats de l'algorithme glouton sont traduits dans le DAG avec un nœud d'égalité (=) associé au noeud u et aux nœuds redondants (fig. 4b).

[1]Si nous ne voulons pas gérer d'expressions conflictuelles, nous pouvons ajouter immédiatement les CS dans le DAG.

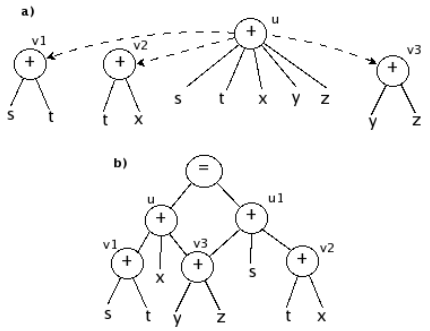

FIG. 4 – Intégration des nœuds d'intersection dans un nœud u. a) Le nœud u a trois CS en conflit. b) Le DAG obtenu après l'utilisation de l'algorithme glouton. Un nœud égalité a été créé pour intégrer les nœuds redondants dans le DAG.

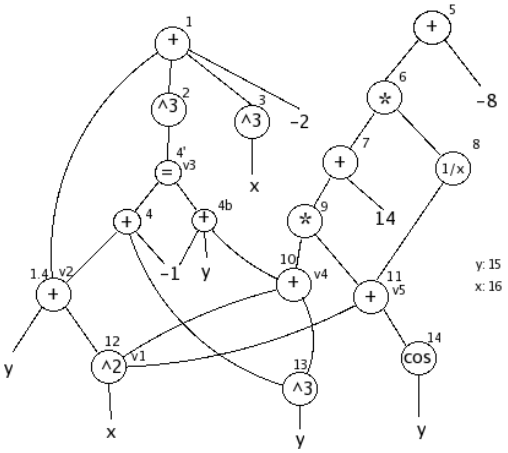

FIG. 5 – DAG obtenu après l'étape 3 : tous les arcs d'inclusion ont été intégrés dans le DAG. Pour les sous-expressions conflictuelles (nœuds 1.4 et 10) un nœud redondant (4b) et un nœud égalité (4') ont été créés. Le nœud 1.4 est lié au nœud 4, et le nœud 10 au nœud 4b. Les variables (v_1, v_2,...) sont générées à l'étape 4, et correspondent aux CS utiles (v_1, v_2, v_4 et v_5), et aux nœuds d'égalité (v_3).

4.4 Étape 4 : Génération du nouveau système

La première manière d'exploiter les CS pour la résolution des NCSP est d'utiliser directement le DAG obtenu après l'étape 3. Comme l'ont montré Vu et al. dans [17], la phase de propagation ne peut plus être effectuée par `HC4` pur, et un algorithme de propagation plus sophistiqué doit examiner le DAG unique correspondant à l'ensemble du système.

Afin de pouvoir continuer à utiliser `HC4` pour la propagation et donc être compatible avec les résolveurs par intervalles existants, nous générons, récursivement, un nouveau système d'(in)équations dans lequel une variable auxiliaire v et une équation $v = f$ sont ajoutées pour chaque CS utile. En évitant de créer de nouvelles équations pour les CS inutiles qui

ne peuvent fournir une contraction additionnelle - cf. section 3, nous réduisons la taille du nouveau système.

En outre, les expressions redondantes $(f, f_1, f_2, ..f_k)$, liées par un nœud d'égalité, ajoutent une nouvelle variable v' dans le système et les équations $v' = f$, $v' = f_1$, ... $v' = f_k$.

L'étape 4 effectue un parcours ascendant du DAG obtenu à l'étape 3 et génère les variables et les équations en chaque nœud.

Finalement, le nouveau système est composé par les équations modifiées, par les variables auxiliaires et par les nouvelles contraintes correspondant aux CS. Le nouveau système correspondant à l'exemple est le suivant :

$$\begin{aligned} v_2 + (v_3)^3 + x^3 - 2 &= 0 \\ \frac{v_4 \times v_5 + 14}{v_5} - 8 &= 0 \end{aligned}$$

$$\begin{array}{lllllll} v_1 & = & x^2 & & v_3 & = & -1 + y + v_4 \\ v_2 & = & y + v_1 & & v_4 & = & v_1 + y^3 \\ v_3 & = & v_2 + y^3 - 1 & & v_5 & = & v_1 + cos(y) \end{array}$$

Pour un système d'équations donné, notre résolveur par intervalles gère deux systèmes : le nouveau système généré par `I-CSE` est utilisé seulement pour `HC4` et le système original est utilisé pour les autres opérations (bissections, Newton). Les intervalles dans les deux systèmes doivent être synchronisés au cours de la recherche de solutions. Tout d'abord, cela nous permet de valider l'intérêt de `I-CSE` pour `HC4`. De plus, envisager des opérations de bissection ou de Newton sur les variables auxiliaires (CS) mériterait une étude approfondie et une validation théorique et pratique. Enfin, ce choix est similaire à l'algorithme de résolution basé sur le DAG proposé par Vu et al. qui prend en compte seulement les variables originales pour les bissections et Newton ; les nœuds internes, correspondant aux CS, sont uniquement utilisés pour la propagation. Leur approche ne gère qu'un système mais un algorithme de propagation sophistiqué doit être utilisé à la place de HC4 [17].

5 Implantation de `I-CSE`

`I-CSE` a été mis en œuvre avec la version 6 de Mathematica. D'abord, Mathematica met automatiquement les équations sous une forme canonique, où les additions et multiplications sont n-aires et où sont effectuées des réductions, c'est-à-dire, des factorisations par une constante. Par exemple, l'expression $2x - y + x + z$ est transformée en $+(\times(3, x), -y, z)$. La représentation n-aire des équations est utile pour l'intersection des expressions deux à deux (étape 2).

Les algorithmes de résolution sont développés dans la bibliothèque par intervalles libre en `C++ Ibex` [3]. Une instance donnée est résolue par un processus *branch and prune* : les variables sont bissectées à tour de rôle et contractées avec des opérateurs de programmation par contraintes (`HC4` seul, ou `3BCID`[2] utilisant `HC4`) et d'analyse par intervalles (Newton). Comme mentionné ci-dessus, `Ibex` offre des facilités pour créer deux systèmes d'équations en mémoire pour lesquels les domaines des variables sont synchronisés au cours de la recherche de solutions. Un système contient la configuration initiale du système d'équations et est utilisé pour la bissection et Newton. L'autre système contient le nouveau système généré par `I-CSE` et est traité par `HC4` (ou `3BCID` utilisant `HC4`).

`I-CSE-B` et `I-CSE-NC`

Nous avons prouvé théoriquement que l'intérêt de `I-CSE` réside dans le filtrage additionnel qu'il permet et non seulement dans la diminution du nombre d'opérations. Pour confirmer dans la pratique cet important résultat, nous avons conçu deux variantes de `I-CSE` qui calculent moins de CS.

`I-CSE-B` (Basic `I-CSE`) ignore simplement l'étape 2 de `I-CSE`. La commutativité et l'associativité de $+$ et $\times$ ne sont pas prises en compte. Les expressions n-aires d'addition et de multiplication sont considérées dans une forme binaire fixée, dans laquelle seulement quelques sous-expressions peuvent être détectées. Par exemple, $x + y$ est détecté dans deux expressions $x + y + z_1$ et $x + y + z_2$, mais pas dans les expressions $x + z_1 + y$ et $x + z_2 + y$.

`I-CSE-NC` (`I-CSE` sans conflits) exploite complètement la commutativité et l'associativité de $+$ et $\times$, mais ne prend pas en compte les CS conflictuelles. `I-CSE-NC` réduit le surcoût éventuel dans le pire des cas, mais ne remplace pas toutes les CS. Le nombre des équations redondantes correspondant aux CS conflictuelle dans le pire des cas est en effet $O(k^2)$, où k est le nombre de nœuds créés à l'étape 1. Si un système donné ne contient pas de CS en conflit, `I-CSE` et `I-CSE-NC` retournent le même nouveau système (sans équations redondantes). Dans l'exemple, `I-CSE-NC` ne crée pas l'équation redondante $v_3 = y + v_7$, ni l'équation $v_7 = v_9 + y^3$. La deuxième équation devient finalement : $\frac{(x^2+y^3)\times v_8+14}{v_8} = 8$.

Les algorithmes CSE existants prennent place entre `I-CSE-B` et `I-CSE-NC` en termes de nombre de CS utiles détectées. Nous supposons ici que l'algorithme proposé par Vu et al. [17] est similaire à `I-CSE-NC`.

6 Expérimentations

Problèmes sélectionnés

Les instances testées ont été sélectionnées dans les deux premières séries (systèmes polynomiaux et non polynomiaux) de la page Web de l'équipe COPRIN[3].

[2] `3BCID` est une variante de `3B` [16].

[3] `www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html`

Les problèmes choisis répondent à des critères systématiques : tous sont des NCSP avec un nombre fini de solutions isolées (pas d'optimisation) ; toutes les solutions peuvent être trouvées par le système ALIAS [11] dans un temps compris entre une seconde et une heure. Les systèmes choisis sont écrits avec les opérateurs de base suivants : `+`, `-`, $\times$, `/`, `sin`, `cos`, `tan`, `exp`, `log`, `power`. Avec ces critères, nous avons sélectionné 40 problèmes. L'algorithme `I-CSE` ne détecte pas de CS dans 16 d'entre eux.

Il y a également deux autres instances (`Fourbar` et `Dipole2`) pour lesquels aucune résolution ne s'est terminée avant le temps limite d'une heure, ne fournissant aucune indication. Les 22 problèmes indiqués dans les tableaux contiennent au moins une CS et fournissent une validation intéressante de notre recherche. 9 de ces 22 problèmes sont paramétrés : ils peuvent être définis avec un nombre de variables quelconque. Le tableau 1 fournit des informations sur les critères sélectionnés. Quand il n'y a pas de CS conflictuelle, `I-CSE` et `I-CSE-NC` retournent le même nouveau système, le nombre de CS est le même (#cs=cs') et il n'y a pas de contraintes redondantes (#rc=0). Les résultats trouvés par le resolveur par intervalles sont les mêmes.

Pour tous les instances, le temps CPU utilisé par `I-CSE` (et ses variantes) est souvent négligeable et est toujours de moins de 1 seconde.

Remarque. Dans les problèmes marqués avec une étoile (*), les équations n'ont pas été initialement écrites sous forme canonique par `Mathematica` (cf. la section 5). Cela conduit à moins de CS, mais ces CS correspondent à des sous-expressions plus grandes partagées par plus d'expressions, offrant généralement de meilleurs résultats.

Les tableaux 2 et 3 comparent les temps CPU nécessaires à `Ibex` pour résoudre le système initial (Init) et les systèmes générés par `I-CSE-B`, `I-CSE-NC` et `I-CSE`. Le tableau 2 montre les résultats obtenus par une approche *branch and prune* standard avec bissection, Newton et **HC4**. Le tableau 3 montre les résultats obtenus par une approche *branch and prune* avec bissection, Newton et **3BCID** (en utilisant `HC4` comme algorithme de réfutation). Les deux tableaux donnent des temps CPU en secondes obtenus par un processeur 2.40 `GHz Intel Core 2` avec 1 `Gb` de RAM, et le gain correspondant par rapport à la résolution du système original. Le temps limite a été fixé à 3600 secondes. Les tableaux indiquent également le nombre de boîtes générées (#Boîtes) au cours de la recherche. Cela correspond au nombre de nœuds dans l'arbre de recherche et met en lumière le filtrage supplémentaire dû à `I-CSE`. La précision des solutions a été fixée à 10^{-8} pour tous les problèmes. Le paramètre utilisé par `HC4` a été fixé à 1% dans le tableau 2. Les paramètres utilisés par `HC4` et `3BCID` ont été fixés à 10% dans le tableau 3. Nous avons mis à la fin des deux tableaux les résultats correspondant aux problèmes paramétrés. Pour obtenir une comparaison équitable entre les algorithmes, nous avons sélectionné pour les systèmes paramétrés l'exemple avec le plus grand nombre de variables n tel que le résolveur sur le système d'origine trouve les solutions en moins d'une heure[4]. Ce nombre n est plus grand avec `3BCID` (tableau 3) que avec `HC4` (tableau 2) parce que `3BCID` est généralement plus efficace que `HC4`.

Intérêt de `I-CSE`

Les tableaux 2 et 3 indiquent clairement que `I-CSE` est très intéressant dans la pratique. Nous constatons un gain de performance supérieur à un facteur 2 sur 15 parmi les 24 lignes (dans les deux tableaux). Le gain est de deux ordres de grandeur (ou plus) pour 5 problèmes avec `HC4` (correspondant à 4 systèmes différents) et pour 10 problèmes avec `3BCID` (correspondant à 8 systèmes différents).

Comparaison entre `I-CSE`, `I-CSE-NC` et `I-CSE-B`

`I-CSE` surpasse clairement les variantes qui extraient moins de CS utiles, comme le montrent le tableau 2 (cf. `Brown-7`, `Dis-Integral-6`, `Broyden-Banded-20`) et le tableau 3 (cf. `Brown-7`, `Dis-Integral-6`, `Yamamura-12`, `Trigonometric-10`). Dans ces cas, les gains en temps CPU sont importants. Ils sont parfois de plusieurs ordres de grandeur. Les quelques exceptions pour lesquelles `I-CSE` est moins performant que sa variante simple donnent seulement un léger avantage à `I-CSE-NC` ou `I-CSE-B`.

Le nombre de boîtes est généralement décroissant de gauche à droite sur les tableaux. Cela confirme notre analyse théorique sur les gains espérés dans le filtrage quand un système a des équations additionnelles en raison de CS.

Cela prouve expérimentalement que l'exploitation de CS conflictuelles est utile. Cela confirme une intuition partagée par un grand nombre de praticiens des algorithmes de cohérence partielle que les contraintes redondantes sont souvent utiles car elles permettent un meilleur effet de filtrage. Les problèmes comme `Brown-30`, `Dis-Integral-20` et `Yamamura-16` ont été ajoutés à la fin du tableau 3 pour mettre en évidence cette tendance : `I-CSE` produit un gain de performance de 3 ordres de grandeur, alors qu'il ajoute des centaines d'équations redondantes.

Quelques résultats mitigés

La plupart des résultats obtenus sont bons ou très bons, mais quatre problèmes obtiennent une perte de performance située entre 20% et 42% : `Caprasse` avec les deux stratégies, et `Pramanik`, `Geneig`, `Katsura` avec `3BCID`.

[4] Notez que lorsque `Trigexp1` est résolu avec `3BCID`, le temps CPU est largement inférieur à 1 heure (cf. le tableau 3).

TAB. 1 – Sélection des instances testées. Les colonnes donnent le nom du problème, le nombre de solutions ($\#s$), le nombre de variables (N), le nombre de CS utiles ($\#cs$) et inutiles (cs') trouvées par `I-CSE-B`, le nombre de CS trouvées par `I-CSE-NC`, le nombre de CS trouvées par `I-CSE`, le nombre de contraintes redondantes créées par `I-CSE` fournies par les CS conflictuelles ($\#rc$).

Problème			I-CSE-B		ICSE-NC	I-CSE			Problème			I-CSE-B		ICSE-NC	I-CSE	
	#s	N	#cs	cs'	#cs	#cs	#rc			#s	N	#cs	cs'	#cs	#cs	#rc
6body	5	6	2	0	3	3	0		Katsura-20	7	21	90	0	90	90	0
Bellido	8	9	0	1	1	1	0		Kin1	16	6	13	0	13	19	3
Brown-7	3	7	3	1	7	21	24		Pramanik	2	8	0	0	15	15	0
Brown-7*	3	7	3	1	1	1	0		Prolog	0	21	0	10	7	7	0
Brown-30	2	30	26	1	53	435	783		Rose	16	3	5	5	5	5	0
BroyBand-20	1	20	22	1	37	97	73		Trigexp1-30	1	30	29	0	29	29	0
BroyBand-100	1	100	102	1	119	479	473		Trigexp1-50	1	50	49	0	49	49	0
Caprasse	18	4	6	2	7	11	2		Trigexp2-11	0	11	15	5	15	15	0
Design	1	9	3	7	3	3	0		Trigexp2-19	0	19	27	9	27	27	0
Dis-Integral-6	1	6	4	14	6	18	9		Trigonom-5	2	5	7	0	9	20	14
Dis-Integral-20	3	20	18	56	34	207	171		Trigonom-5*	2	5	7	0	6	6	0
Eco9	16	8	0	0	3	7	1		Trigonom-10	24	10	15	2	15	26	15
EqCombustion	4	5	7	2	8	11	1		Trigonom-10*	24	10	15	2	12	12	0
ExtendWood-4	3	4	2	2	2	2	0		Yamamura-8	7	8	5	0	10	36	48
Geneig	10	6	11	0	14	14	0		Yamamura-8*	7	8	5	0	1	1	0
Hayes	1	8	9	4	8	8	0		Yamamura-12	9	12	9	0	18	78	119
I5	30	10	3	12	4	10	5		Yamamura-12*	9	12	9	0	1	1	0
Katsura-19	5	20	81	0	81	81	0		Yamamura-16	9	16	13	0	26	136	224

TAB. 2 – Résultats obtenus avec `HC4` et Newton

Problème	**Temps (s)**				**Temps (Init) / Temps**			**#Boites**		
	Init	ICSE-B	ICSE-NC	I-CSE	ICSE-B	ICSE-NC	I-CSE	Init	ICSE-NC	I-CSE
EqCombustion	>3600	29.4	0.47	**0.4**	>120	>7600	**>9000**	>8e+06	6243	**5269**
Rose	>3600	786	**151**	**151**	>4.6	**>24**	**>24**	>2e+07	**881673**	**881673**
Hayes	141	51.9	**15.7**	**15.7**	2.7	**9**	**9**	550489	**44563**	**44563**
6-body	0.22	**0.07**	**0.07**	**0.07**	**3.1**	**3.1**	**3.1**	4985	**495**	**495**
Design	176	65.2	**63.2**	**63.2**	2.7	**2.8**	**2.8**	425153	**122851**	**122851**
I5	>3600	>3600	**1605**	1689	?	**>2.2**	>2.1	>2e+07	**9e+06**	9e+06
Pramanik	89.3	92.1	**84.9**	**84.9**	0.97	**1.05**	**1.05**	487255	**378879**	**378879**
Kin1	8.52	8.32	8.32	**8.01**	1.02	1.02	**1.06**	**905**	909	**905**
Bellido	15.7	15.9	**15.6**	**15.6**	0.99	**1.01**	**1.01**	29759	**29319**	**29319**
Eco9	**23.9**	**23.9**	24	24.1	**1.00**	**1.00**	0.99	126047	117075	**110885**
Caprasse	**1.56**	1.81	1.68	2.16	0.86	0.93	0.72	8521	7793	**7491**
Geneig	>3600	>3600	>3600	>3600	?	?	?	?	?	?
Brown-7*	495	348	**0.01**	**0.01**	1.42	**49500**	**49500**	6e+06	**95**	**95**
Dis-Integral-6	201	0.46	1.3	**0.03**	437	155	**6700**	653035	4157	**47**
ExtendWood-4	29.9	**0.03**	**0.03**	**0.03**	**997**	**997**	**997**	422705	**353**	**353**
Brown-7	495	348	30.7	**1.49**	1.42	16.1	**332**	6e+06	258601	**3681**
Trigexp2-11	1205	238	**59.5**	**59.5**	5.06	**20.3**	**20.3**	2e+06	**348087**	**348087**
Yamamura-8*	13	13.3	**0.75**	**0.75**	0.98	**17.3**	**17.3**	29615	**2161**	**2161**
Broy-Banded-20	1036	1094	391	**61.9**	0.95	2.65	**16.7**	210531	71719	**13619**
Trigonometric-5*	15.8	12.3	**1.49**	**1.49**	1.28	**10.6**	**10.6**	10531	**1503**	**1503**
Trigonometric-5	15.8	12.3	8.94	**6.97**	1.28	1.77	**2.27**	10531	7369	**5307**
Yamamura-8	13	13.3	14.1	**10.8**	0.98	0.92	**1.20**	29615	22710	**13211**
Katsura-19	**1430**	1583	1583	1583	0.90	0.90	0.90	**145839**	153193	153193
Trigexp1-30	**2465**	3244	3244	3244	0.76	0.76	0.76	1e+07	1e+07	1e+07

La perte de performance observée pour `Katsura` (10% ou 42% selon la stratégie) est due aux domaines des variables qui sont initialisés à [0,1]. Sans entrer dans les détails, de tels domaines impliquent que le filtrage dans l'arbre de recherche est dû à la phase d'évaluation et non pas à la phase de contraction de `HC4-revise`.

7 Conclusion

Cet article a présenté l'algorithme `I-CSE` pour exploiter des sous expressions communes dans les NCSP. Une analyse théorique a montré que les gains dans le filtrage peuvent seulement être escomptés quand les CS ne correspondent pas à des opérateurs monotones et continus comme x^3 ou log. Contrairement à ce que l'on croyait auparavant, cela signifie que les CS peuvent apporter des gains importants en contraction, et non seulement une diminution du nombre d'opérations. Ce sont de bonnes nouvelles pour l'importance de cet axe de recherche.

Des expériences ont été effectuées sur 40 problèmes dont 24 contiennent des CS. Des gains importants d'un ou de plusieurs ordres de grandeur ont été observés sur 10 d'entre eux. `I-CSE` diffère des CSE existants dans le fait qu'il exploite aussi des CS conflictuelles. Par rapport à `I-CSE-NC` (similaire aux CSE existants), la contraction additionnelle impliquée par les équations redondantes conduit à l'amélioration d'un ou plusieurs ordres de grandeur sur 4 problèmes (`Brown`, `Dis-Integral`, `Yamamura` et, uniquement pour `HC4`, `Broy-Banded`).

Un travail futur serait de comparer notre application basée sur un algorithme `HC4` standard (et la gestion de deux systèmes), avec l'algorithme de propagation so-

TAB. 3 – Résultats obtenus avec 3BCID (utilisant HC4) et Newton

Problème	Temps (s)				Temps(Init) / Temps			#Boîtes		
	Init	ICSE-B	ICSE-NC	I-CSE	ICSE-B	ICSE-NC	I-CSE	**Init**	ICSE-NC	I-CSE
Rose	2882	5.17	**4.04**	**4.04**	557	**713**	**713**	4e+06	**5711**	**5711**
Prolog	38.5	60	**0.14**	**0.14**	0.64	**275**	**275**	4647	**11**	**11**
EqCombustion	0.42	0.37	**0.06**	**0.06**	1.35	7	7	427	**23**	**23**
Hayes	32.6	27.2	**5.67**	**5.67**	1.13	**5.7**	**5.7**	17455	**1675**	**1675**
Design	52	17.9	**13.3**	**13.3**	2.9	**3.9**	**3.9**	16359	**4401**	**4401**
I5	33.5	41.1	**17.9**	**17.8**	0.81	1.9	**1.9**	10619	4387	**4281**
6-body	0.14	**0.08**	0.1	0.1	**1.75**	1.4	1.4	173	**51**	**51**
Kin1	1.66	2.66	1.76	**1.23**	0.62	0.94	**1.35**	**85**	161	197
Bellido	10.3	10.4	**9.98**	**9.98**	1	**1.03**	**1.03**	4487	**4341**	**4341**
Eco9	**11.6**	**11.6**	12.4	13.2	**1**	0.94	0.88	6205	6045	**5749**
Pramanik	**73.8**	114	96.8	96.8	0.65	0.76	0.76	124663	**95305**	**95305**
Caprasse	1.96	2.51	2.5	2.92	0.74	0.78	0.67	1285	1311	**1219**
Geneig	**696**	1050	1050	1050	0.66	0.66	0.66	362225	**362045**	**362045**
Trigexp2-19	2308	2.23	**0.03**	**0.03**	1035	**77000**	**77000**	250178	**7**	**7**
Brown-7*	600	318	**0.01**	**0.01**	1.88	**60000**	**60000**	662415	**9**	**9**
ExtendWood-4	185	**0.03**	**0.03**	**0.03**	6167	6167	6167	669485	**35**	**35**
Dis-Integral-6	135	0.18	0.51	**0.03**	750	264	**4500**	86487	185	**7**
Brown-7	600	318	4.75	**0.22**	1.88	126	**2700**	662415	2035	**23**
Yamamura-12*	1751	1842	**1.01**	**1.01**	0.95	**1700**	**1700**	364105	**307**	**307**
Yamamura-12	1751	1842	31.1	**8.72**	0.95	56.3	**200**	364105	5647	**445**
Trigono-10*	1344	506	**19.4**	**19.4**	2.67	**69**	**69**	140512	**2033**	**2033**
Trigono-10	1344	506	156	**49.6**	2.67	8.62	**27**	140512	19883	**3339**
Broy-Banded-100	9.96	20.3	14.8	**8.21**	0.49	0.67	**1.21**	13	23	**11**
Trigexp1-50	**0.15**	0.19	0.17	0.17	0.79	0.88	0.88	**1**	**1**	**1**
Katsura20	**3457**	5919	5919	5919	0.58	0.58	0.58	**62451**	120929	120929
Brown-30	>3600	>3600	>3600	**22.9**	?	?	>150	>210021	>151527	**31**
Dis-Integral-20	>3600	>3600	>3600	**1.12**	?	?	**>3200**	>111512	>75640	**39**
Yamamura-16	>3600	>3600	681	**35.6**	?	>5	**>100**	>522300	96341	**919**

phistiqué effectué sur l'élégante structure de DAG proposée par Vu, Schichl et Sam-Haroud. Toutefois, nos résultats expérimentaux ont souligné que le gain en contraction a une plus grande influence sur l'efficacité que le temps nécessaire pour atteindre le point fixe de la propagation. Ainsi, nous pensons que les deux implémentations afficheraient des performances similaires.

Références

[1] F. Benhamou, F. Goualard, L. Granvilliers, and J-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, pages 230–244, 1999.

[2] D. P. Brown. *Calculus and Mathematica*. Addison-Wesley, 1991.

[3] G. Chabert. `www.ibex-lib.org`, 2008.

[4] H. Collavizza, F. Delobel, and M. Rueher. Comparing partial consistencies. *Reliable Computing*, 5(3) :213–228, 1999.

[5] R. Debruyne and C. Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proc. IJCAI*, pages 412–417, 1997.

[6] P. Flajolet, P. Sipala, and J-M. Steyaert. Analytic variations on the common subexpression problem. In *Proc. Automata, Languages, and Programming, LNCS 443*, pages 220–334, 1990.

[7] L. Granvilliers, E. Monfroy, and F. Benhamou. Symbolic-Interval Cooperation in Constraint Programming. In *Proc. ISSAC, ACM*, pages 150–166, 2001.

[8] A. Heck. *Introduction to Maple*. Springer-Verlag, 2003.

[9] Y. Lebbah. *Contribution à la Résolution de Contraintes par Consistance Forte*. Phd thesis, Université de Nantes, 1999.

[10] O. Lhomme. Consistency Tech. for Numeric CSPs. In *IJCAI*, pages 232–238, 1993.

[11] J-P. Merlet. ALIAS : An Algorithms Library for Interval Analysis for Equation Systems. Technical report, INRIA Sophia, 2000. `www-sop.inria.fr/coprin/logiciels/ALIAS/ALIAS.html`.

[12] J-P. Merlet. Interval Analysis and Robotics. In *Symp. of Robotics Research*, 2007.

[13] S. Muchnick. *Advanced Compiler Design and Implem.* M. Kauffmann, 1997.

[14] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[15] H. Schichl and A. Neumaier. Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization*, 33(4) :541–562, 2005.

[16] G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP'07, LNCS 4741*, pages 635–650, 2007.

[17] X.-H. Vu, H. Schichl, and D. Sam-Haroud. Using Directed Acyclic Graphs to Coordinate Propagation and Search for Numerical Constraint Satisfaction Problems. In *Proc. ICTAI'04, IEEE*, pages 72–81, 2004.