Améliorer la propagation : l'Importance d'être Inconsistant

Ghiles Ziat, Marie Pelleau, Charlotte Truchet, Antoine Miné

▶ **To cite this version:**

Ghiles Ziat, Marie Pelleau, Charlotte Truchet, Antoine Miné. Améliorer la propagation : l'Importance d'être Inconsistant. Treizièmes journées Francophones de Programmation par Contraintes, Jun 2017, Montreuil sur Mer, France. hal-01735167

HAL Id: hal-01735167
https://hal.science/hal-01735167

Submitted on 15 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Améliorer la propagation : l'Importance d'être Inconsistant

Ghiles Ziat[1*] **Marie Pelleau**[1] **Charlotte Truchet**[2] **Antoine Miné**[1]

[1] Sorbonne Universités, UPMC Univ. Paris 06, CNRS, LIP6, Paris, France
[2] Université de Nantes, France
{prénom.nom}@lip6.fr charlotte.truchet@univ-nantes.fr

Résumé

Les méthodes classiques de résolution de problèmes de satisfaction de contraintes alternent généralement deux étapes : la propagation et l'exploration. La propagation réduit les domaines en fonction des contraintes. Elle peut être vue comme une étape de division de l'espace de recherche en deux parties : le sous-espace inconsistant, qui est éliminé du processus de résolution et le sous-espace indéterminé qui contient les solutions du problème. L'étape d'exploration consiste alors à diviser le sous-espace indéterminé en plusieurs sous-espaces où continuer la résolution. Cette étape est implantée dans la plupart des solveurs de contraintes à l'aide d'heuristiques de coupe qui reposent sur les domaines des variables et/ou les contraintes du problème. Cet article introduit une nouvelle étape dans la résolution appellée *élimination*. Elle divise l'espace de recherche en deux sous-espaces : l'espace indéterminé et l'espace consistant. Elle permet de mieux tirer profit des contraintes et ainsi d'obtenir des frontières plus pertinentes pour la résolution. Cette nouvelle étape est basée sur une observation clé : la partie consistante d'un problème est équivalente à la partie inconsistante du problème complémentaire. Nous avons implanté cette méthode au sein du solveur continu AbSolute. Ce solveur mixe des méthodes d'Interpretation Abstraite et de Programmation Par Contraintes, et la technique d'élimination s'y intègre bien. Les premiers résultats expérimentaux montrent des améliorations significatives des performances du processus de résolution.

Abstract

Classical CSP solving methods often alternate two steps : propagation and exploration. Propagation reduces the domains of the variables according to the constraints. It can be seen as a discrimination of the search space in two sub-spaces : the inconsistent one that can be deleted from the solving process, and the undetermined one which may contain the solutions. The exploration step divides the undetermined sub-space into several sub-spaces in which the search continues. This step is usually implemented in solvers by split heuristics relying onto the domains of the variables and/or the constraints of the problem. This article introduces a new step into the solving process called *elimination*. It divides the search space into two sub-spaces : the undetermined one and the consistent one. It allows the solver to benefit more from the constraints, thus obtaining more significative frontiers for the exploration. It is based on a key observation : the consistent part of a problem is equivalent to the inconsistent part of the complementary problem. This new step is implemented in the AbSolute continuous solver. This solver combines methods from Abstract Interpretation and Constraint Programming. Our elimination technique can be easily added in it. Preliminary results show significative improvements of the solving process.

1 Introduction

La résolution de CSP implique génréralement l'alternance de deux étapes : la propagation et l'exploration. La propagation réduit les domaines en fonction des contraintes. L'exploration quant à elle ajoute des hypothèses afin de diviser le problème en plusieurs sous-problèmes plus faciles à résoudre. Dans le continu, la méthode de résolution peut retourner un pavage de l'espace de recherche à l'aide d'éléments simples à manipuler (des boîtes, en général). Ce pavage peut correspondre à une approximation extérieures ou sur-approximation des solutions, comme dans le solveur Ibex [7], ou peut correspondre à une approximation intérieure ou sous-approximation [8].

Que ce soit dans le discret ou dans le continu, il n'existe pas une unique façon d'explorer l'espace de recherche, et différentes heuristiques existent. Ces heu-

(a) Sous-approximation (b) Sur-approximation

Figure 1 – Comparaison entre la sous-approximation et la sur-approximation.

ristiques utilisent généralement la taille des domaines et/ou les contraintes afin de déterminer quelle variable instancier ou quel domaine couper. Dans le discret, les heuristiques les plus connues sont *dom* ou *fail-first* [13] choisissant la variable avec le plus petit domaine, et *dom* + *deg* [6], *dom* / *deg* [4] et *dom* / *w deg* [5] choisissant, toutes les trois, la variable en fonction de la taille de son domaine et du nombre de contraintes dans lesquelles elle apparaît. Dans le continu, les heuristiques classiques sont *largest first* [19] choisissant le domaine de plus grande taille à couper, *round robin*, où les domaines sont traités successivement, ou encore *maximal smear* [12] basée sur les dérivées des contraintes et choisissant le domaine avec la plus grande pente. Plus récemment, *Mind The Gaps* [1] reprend l'idée de [12, 19] et utilise des consistances partielles pour couper les domaines.

Dans cet article, nous proposons d'ajouter une nouvelle étape à la méthode de résolution continue : l'élimination. Cette phase divise l'espace de recherche en deux sous-espaces : le premier contenant uniquement des solutions, et le second pouvant contenir des solutions. Notre méthode de résolution alterne donc trois étapes, la propagation, l'élimination et l'exploration. Notre nouvelle étape s'inspire des contracteurs utilisés dans Ibex ([7]) pour réaliser une exploration plus intelligente, de façon assez similaire à *Mind The Gaps* [1]. Afin de s'abstraire de la représentation par intervalles nous ajouterons notre nouvelle étape à la méthode de résolution abstraite présentée dans [17]. Notons que cette méthode permet de calculer simultanément une approximation extérieure et intérieure de l'ensemble des solutions.

Cet article est organisé comme suit : nous rappelons dans la section 2 les notions préliminaires d'intrepretation abstraite nécéssaires à la compréhension de la méthode de résolution présentée dans [17]. Nous présentons cette méthode dans la section 3. La section 4 introduit l'étape d'élimination. Nous nous intéresserons aux performances dans la section 5. La section 6 présente les travaux futurs et la conclusion.

2 Rappels

L'interprétation abstraite (IntAbs) [9] est une théorie générale de l'approximation de la sémantique des systèmes discrets dynamiques. Sa principale application est l'analyse statique de programmes, basée sur la sémantique. Elle remplace les calculs trop coûteux de la sémantique réelle par des calculs plus simples, bien que moins précis, réalisés dans un domaine abstrait qui écarte les détails non pertinents. De nombreux domaines abstraits ont été proposés, réalisant divers compromis coût/précision et adaptés à différents types d'analyses. En particulier, pour les propriétés numériques, l'IntAbs propose des domaines abstraits numériques [14], qui peuvent représenter et manipuler efficacement des ensembles de points dans un espace vectoriel représentant une sur-approximation des états de programme accessibles. Un élément clé de l'IntAbs est la dérivation systématique d'une analyse abstraite à partir de la sémantique concrète par application d'une opération d'abstraction. Cette abstraction aboutit à une analyse statique qui est correcte par construction.

3 Résolution abstraite

En PPC, les problèmes de satisfaction de contraintes (CSP) sont modélisés à l'aide de triplets $(\mathcal{X}, \mathcal{D}, \mathcal{C})$:

— $\mathcal{X} = \{x_1, ..., x_n\}$, les variables du problème

— $\mathcal{D} = \{d_1, ..., d_n\}$, les domaines de ces variables tels que $x_k \in d_k, \forall k \in [1, n]$

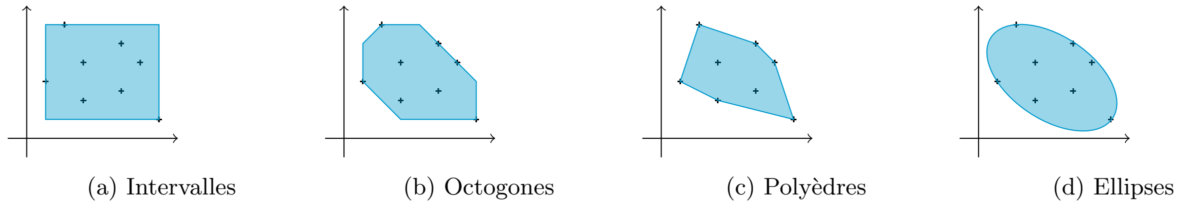
— $\mathcal{C} = \{c_1, ..., c_m\}$, les contraintes du problème

où n est le nombre de variables et m le nombre de contraintes.

Définition 1 (Solution concrète)**.** Une solution d'un CSP est une instanciation de toutes les variables qui respecte toutes les contraintes.

3.1 Un peu d'abstraction

On rappelle ici les notions importantes permettant de définir la méthode de résolution présentée dans [17]. Résoudre un CSP revient à calculer l'ensemble S de ses solutions. Celui-ci pouvant être trop coûteux à calculer



(a) Intervalles (b) Octogones (c) Polyèdres (d) Ellipses

FIGURE 2 – Abstraction d'un ensemble de points avec différents domaines abstraits.

(ou impossible à calculer exactement dans le continu), il peut être plus intéressant d'en trouver une approximation. Nous cherchons ainsi à manipuler un ensemble d'instanciations, représentées par une propriété particulière, plutôt que de les manipuler une à une. Cela se traduit par la construction d'un recouvrement de l'espace de recherche à l'aide d'une disjonction d'éléments abstraits (intervalles, octogones ...).

Définition 2 (Solution abstraite)**.** Soit S l'ensemble des solutions concrètes, on calcule :

— une sur-approximation (ou approximation extérieure) $S^\sharp$ de S telle que : $\forall s \in S \Rightarrow s \in S^\sharp$

— une sous-approximation (ou approximation intérieure) $S^\sharp$ de S telle que : $\forall s \in S^\sharp \Rightarrow s \in S$

Une méthode de résolution est dite *complète* si elle retourne une sur-approximation, et est dite *correcte* si elle retourne une sous-approximation de l'ensemble des solutions. En continu, les méthodes de résolution sont généralement complètes.

La figure 1 compare sur un même exemple le résultat obtenu par sous-approximation (1a) et celui obtenu par sur-approximation (1b). Dans cet exemple on cherche à approximer une zone (en bleu), on peut voir que les boîtes solutions dans la figure 1a sont totalement incluses dans la zone, elles ne contiennent que des solutions. En revanche, dans la figure 1b, certaines boîtes ne contiennent pas que des solutions, on a une sur-approximation de l'espace des solutions.

Dans les deux cas, on peut juger de la qualité de la couverture produite par un solveur selon plusieurs critères. Nous déciderons dans ce travail de se concentrer sur deux critères :

— Le nombre d'éléments : les solveurs étant souvent utilisés comme première étape d'un traitement spécifique, nous essaierons ici de minimiser le nombre d'éléments dans la couverture de l'espace, pour faciliter la réutilisation des résultats du solveur. Les avantages liés à cette réduction peuvent être important dans le cadre d'applications d'approximations intérieures où il est plus intéressant de recouvrir l'espace de peu de "gros" morceaux plutôt que de beaucoup de petits.

— La redondance : si les résultats obtenus ne s'intersectent pas entre eux, alors on dit de la solution qu'elle est non-redondante. Cette propriété garantit qu'on ne traite pas plusieurs fois les mêmes instanciations concrètes et peut être souhaitable dans le cadre d'application à la peinture industrielle, (ne pas peindre plusieurs fois une même surface), impression 3D, etc.

La méthode de résolution que nous présentons ici est aussi bien adaptée à une résolution complète que correcte. Elle permet, avec la majorité des domaines abstraits utilisés, d'être non-redondante. Aussi, la technique d'élimination permet sur beaucoup d'exemples de réduire le nombre d'éléments de la couverture.

3.2 Domaines abstraits

La méthode de résolution définie dans [17] se définit en transposant les concepts de propagation et d'exploration aux domaines abstraits. Pour approximer l'espace de solution, elle utilise des opérations sur un nombre fini d'éléments abstraits. Cette méthode est paramétrée par un domaine abstrait qui redéfinit les concepts usuels de programmation par contraintes.

Les domaines abstraits sont fondés sur la notion de treillis et permettent d'encoder un certain type d'approximation d'états. Un treillis est un ensemble muni d'un ordre partiel (relation réflexive, transitive et antisymétrique). Les domaines abstraits doivent de plus implanter pusieurs opérations nécéssaires à leurs manipulations (union, intersection, élargissement ...). Ils sont très largement utilisés en analyse de programmes [9, 3] et plusieurs domaines, proposant différents compromis coût/précision ont été développés.

Le domaine des intervalles, par exemple, permet d'encoder des relations de la forme : $\bigwedge i \in 1..n,\ a_i < x_i < b_i$ où n est le nombre de dimensions de l'espace, et a_i et b_i sont des constantes. Le domaine des octogones [16] est plus expressif et autorise des relations de la forme $\pm x_i \pm x_j$ entre les variables.

La figure 2 montre l'abstraction d'un même espace avec quatre domaines abstraits parmi les plus utilisés : les intervalles [9], les octogones, les polyèdres [10] et les ellipsoïdes [11].

Nous donnons ici la définition des domaines abstraits tels qu'utilisés dans notre méthode de résolution :

Définition 3 (Domaine abstrait)**.** Un domaine abstrait est défini par :
- un ordre partiel $(E, \subseteq)$, (où les éléments de E sont représentables sur machine),
- une fonction d'abstraction α du domaine des variables vers E,
- une fonction de concretisation γ qui forme une correspondance de Galois avec α,
- un plus petit élément $\bot$,
- un plus grand élément $\top$.

Cette définition classique, fondée sur un ordre partiel, permet d'établir une hiérarchie entre éléments abstraits. Les domaines abstraits que nous utiliserons doivent implanter plusieurs fonctions spécifiques à la résolution de CSP : Nous ajoutons à la définition des domaines abstraits les éléments suivants :

Définition 4 (Domaines abstraits pour les contraintes)**.**
- un opérateur de consistance : $cons_E : E^\sharp \to c \to E^\sharp$ (avec c une contrainte du problème),
- un opérateur de coupe $\oplus_E : E^\sharp \to E^\sharp \times \cdots \times E^\sharp$,
- une fonction de taille $\tau_E : E^\sharp \to \mathbb{R}$.

L'ajout d'un opérateur de consistance est nécéssaire pour l'étape de propagation, tandis que l'opérateur de coupe permet de définir une étape d'exploration. Ces opérateurs étant décrits précisément dans [17] et [18], nous ne les décrirons pas dans le détail. Voici cependant un rappel de leur usage. L'opérateur de consistance permet de filtrer les valeurs inconsistantes des éléments abstraits. L'ajout d'une fonction de taille nous permet de définir un critère de terminaison. En effet, l'exploration d'une branche s'arrêtera lorsqu'un élément abstrait de celle-ci est plus petit qu'une certaine valeur fixée à l'avance.

L'opérateur de coupe permet de continuer la recherche dans un élément après l'étape de propagation : lorsqu'un élément ne satisfait pas toutes les contraintes, il est divisé en plusieurs sous-éléments. Nous ne détaillerons pas les différentes heuristiques de coupe d'un élément dans cet article, et utiliseront uniquement la technique de coupe *largest first* qui divise chaque élément en deux, dans un axe perpendiculaire à la plus longue dimension. Ainsi on peut redéfinir une méthode de résolution reprenant les principes de propagation-exploration et basée sur les domaines abstraits. L'algorithme 1 donne le pseudo-code de cette méthode de résolution abstraite.

L'algorithme construit une disjonction d'éléments abstraits. À chaque étape, l'élément courant est filtré par rapport aux contraintes à l'aide de l'opérateur de consistance. S'il ne satisfait pas toutes les contraintes, il est divisé à l'aide de l'opérateur de coupe et on propage les contraintes dans les éléments résultants. On repète récursivement ces étapes jusqu'à ce que les éléments obtenus satisfassent les contraintes ou soient plus petits qu'un paramètre r de l'algorithme. Si un élément satisfait toutes les contraintes, on l'ajoute directement à l'ensemble des solutions intérieures retourné par l'algorithme. Si un élément ne satisfait pas toutes les contraintes et qu'il est trop petit pour être divisé, il est ajouté à la liste des solutions si l'on désire avoir une résolution complète, ou ignoré si l'on désire avec une résolution correcte.

Algorithme 1 Résolution paramétrée par un domaine abstrait A

```
function SOLVE(D, C, r)          ▷ D : domaines, C : contraintes
    sols ← ∅                      ▷ solutions trouvées
    explore ← ∅                   ▷ éléments à explorer

    e = init(D)                   ▷ initialisation

    push e in explore

    while explore ≠ ∅ do
        e ← pop(explore)
        e ← filtre(e, C)
        if e ≠ ∅ then
            if τ_A(e) ≤ r or satisfait(e, C) then
                sols ← sols ∪ e
            else
                push ⊕_A(e) in explore
            end if
        end if
    end while
end function
```

Les trois fonctions `init`, `filtre` et `satisfait` sont génériques et ne dépendent pas de la représentation du domaine abstrait. La fonction `init` crée un élément abstrait à partir des domaines initiaux du problème. La fonction `filtre` correspond à la boucle de propagation, elle applique la consistance ($cons_A$) pour chacune des contraintes. La fonction `satisfait` quant à elle vérifie si un élément abstrait est une solution pour toutes les contraintes, c'est-à-dire si il ne contient que des solutions. Cette fonction correspond à un contracteur tel que défini dans [7]. Ces fonctions se déduisent facilement de la consistance utilisée et sont données par les pseudos-codes 3, 4 et 5 en annexe.

La figure 3 montre le résultat de l'algorithme 1 pour le CSP suivant :

$$x_1 \in [-4, 10]$$

$$x_2 \in [0, 5]$$

$$x_2 \leq x_1^2$$

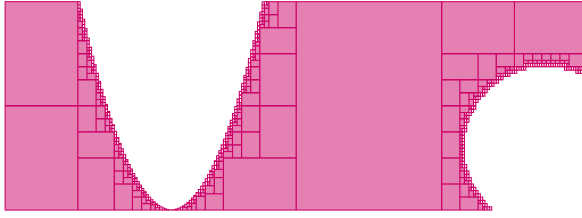


FIGURE 3 – 229 éléments intérieurs, 310 éléments frontaliers.

$$(x_1 - 9)^2 + (x_2 - 1.5)^2 > 4$$

Il est intéressant de noter ici que cette méthode de résolution permet à la fois de sous-approximer l'espace de recherche en tenant compte uniquement des éléments intérieurs, ou de le sur-approximer en tenant compte de tous les éléments retournés. On peut alors s'intéresser à améliorer la couverture produite par le solveur. On peut voir sur la figure 3 que certains des éléments intérieurs pouraient être fusionnés afin d'obtenir de plus grandes boites intérieures. Cette observation traduit le fait que certaines étapes d'exploration sont inutiles. L'étape d'élimination part de cette observation et, en utilisant les contraintes, va essayer de trouver de plus grandes boîtes intérieures.

4 Élimination

L'étape de propagation permet de réduire l'espace de recherche en y retirant des sous-espaces non consistants. Si elle permet alors de sur-approximer l'espace des solutions, on peut définir un symétrique de cette opération pour guider la résolution. Dans cette approche, nous nous intéressons à l'ensemble des instanciations qui ne peuvent être des solutions. Par élimination, le reste de l'espace de recherche ne peut contenir que des solutions. Plus précisément, on cherche à sur-approximer l'ensemble des instanciations qui ne satisfont pas au moins une contrainte. Nous nous référerons par la suite à ces instanciations comme les instanciations inconsistantes ou non-consistantes.

Définition 5 (Complémentaire)**.** Soit $\overline{S}$ le complémentaire des solutions du CSP, c'est-à-dire l'ensemble des instanciations qui ne sont pas solutions.

Cet espace pouvant être incalculable, nous en calculons une sur-approximation $\overline{S}^\sharp$. Cette étape est directe car calculer $\overline{S}^\sharp$ revient à faire une étape de propagation sur la négation des contraintes du CSP.

Nous pouvons alors distinguer dans l'espace de recherche deux sous-espaces :

- l'ensemble des instanciations consistantes inclus dans $S^\sharp$
- l'ensemble des instanciations non-consistantes inclus dans $\overline{S}^\sharp$

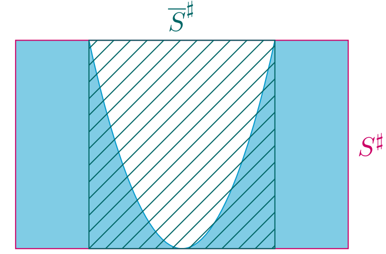


FIGURE 4 – Étant donnée la contrainte en bleu, la boîte $S^\sharp$ sur-approxime les solutions et la boîte hachurée $\overline{S}^\sharp$ sur-approxime les inconsistances.

La figure 4 donne un exemple du complémentaire pour une contrainte donnée. Selon la contrainte en bleu, la boîte $S^\sharp$ (en rose) sur-approxime les solutions et la boîte $\overline{S}^\sharp$ (hachurée en vert) sur-approxime les inconsistances.

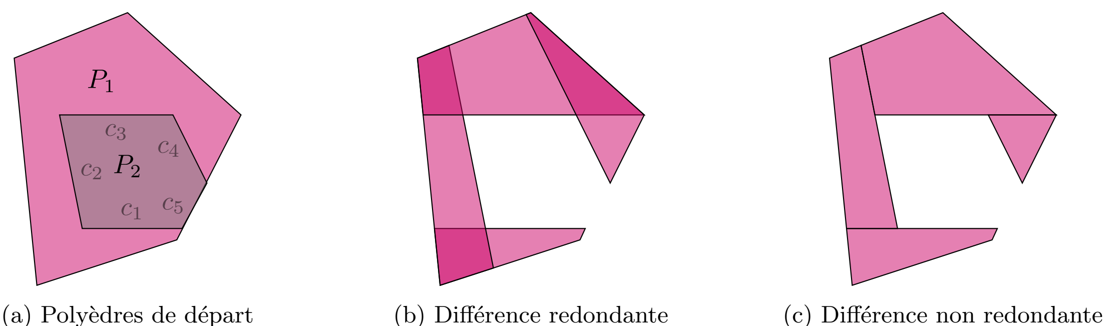
Nous pouvons remarquer que $S^\sharp \setminus \overline{S}^\sharp$ ne contient que des solutions. De plus, $S^\sharp$ peut se définir comme $(S^\sharp \setminus \overline{S}^\sharp) \cup (S^\sharp \cap \overline{S}^\sharp)$. Nous nous baserons sur cette observation pour proposer une amélioration de notre méthode de résolution : nous pouvons directement récupérer de notre espace de recherche les valeurs $(S^\sharp \setminus \overline{S}^\sharp)$, celles-ci étant consistantes par définition, et continuer la résolution dans $(S^\sharp \cap \overline{S}^\sharp)$.

Cette méthode recquiert donc de calculer les espaces $S^\sharp \cap \overline{S}^\sharp$ et $S^\sharp \setminus \overline{S}^\sharp$. Elle n'est donc possible que si les domaines ont un opérateur d'intersection et un opérateur de différence $\setminus$. La plupart des domaines abstraits sont clos par intersection et proposent des opérateurs d'intersection exacts ; calculer $S^\sharp \cap \overline{S}^\sharp$ ne pose aucun problème. Pour calculer $S^\sharp \setminus \overline{S}^\sharp$, il est nécessaire de définir un opérateur de différence.

4.1 Opérateur de différence

Dans cette section, nous cherchons à séparer un élément abstrait e_1 en deux sous-parties : une partie dite consistante, qui satisfait toutes les contraintes du problème et une partie dite non-consistante, qui ne satisfait pas au moins une d'entre elle. Ces deux sous-parties devront recouvrir entièrement e_1. Nous determinerons ces deux parties à l'aide d'un second élément e_2 qui sur-approxime la partie non-consistante du problème. Nous pouvons alors séparer e_1 en deux parties : $e_1 - e_2$ et $e_1 \cap e_2$. Nous travaillerons ici avec les équivalents abstraits de ces deux valeurs.

Définition 6 (Opérateur de différence)**.** Un opérateur



(a) Polyèdres de départ (b) Différence redondante (c) Différence non redondante

FIGURE 5 – Comparaison du résultat de l'opérateur de différence : $P_1 \ominus P_2$

de différence $\ominus : \mathcal{D}^\sharp \times \mathcal{D}^\sharp \to \wp(\mathcal{D}^\sharp)$ est un opérateur binaire tel que $\forall e_1, e_2 \in \mathcal{D}^\sharp$:

1. $|e_1 \ominus e_2|$ est finie
2. $e_i \in (e_1 \ominus e_2) \Rightarrow e_i \cap e_2 = \emptyset$
3. $\gamma(e_1) = \gamma(e_1 \cap e_2) \cup \{\gamma(e_i) \,|\, e_i \in (e_1 \ominus e_2)\}$

Grâce à cet opérateur, nous pouvons ajouter directement le résultat de $e_1 \ominus e_2$ à l'ensemble des solutions de la résolution, et continuer la recherche dans $e1 \cap e2$.

La première condition permet de s'assurer que la résolution produit un ensemble fini de solutions. La deuxième s'assure que l'opérateur est correct, en effet seules les solutions sont traitées comme des solutions. Finalement, la troisième condition garantit la complétude de la résolution, aucune solution n'est perdue.

En pratique, l'utilisation de domaines abstraits convexes classiques (e.g., conjonctions de contraintes linéaires), à même de représenter la négation de contraintes, permet l'implantation d'un opérateur de différence exact.

4.1.1 Pour les domaines convexes

Plusieurs représentations des domaines convexes sont possibles. Dans [17], la représentation par générateurs est utilisée pour calculer les frontières de l'opérateur de coupe tandis que la représentation sous forme de contraintes est préférée pour les calculs de consistance. Nous utiliserons cette seconde représentation dans cette section. Les domaines abstraits les plus populaires (intervalles, octogones, polyèdres) peuvent être définis comme une conjonction de contraintes linéaires. En utilisant cette représentation, l'opérateur de différence est le même pour les polyèdres, les octogones et les intervalles.

Un polyèdre (resp. octogone, intervalle) peut être défini par $\mathcal{P} = \{c_1, ..., c_p\}$, où $c_i : \sum a_i \times x_i \lhd b_i$, avec $\lhd \in \{<, \leq\}$. Notons ici qu'il est necessaire de pouvoir exprimer des contraintes strictes et des contraintes larges afin de pouvoir exprimer exactement la négation des contraintes.

On s'intéresse alors à la définition de la différence de deux polyèdres $P_1 \setminus P_2$: on cherche à identifier du polyèdre P_1 ses sous-parties qui n'intersectent pas P_2.

Nous définissons dans un premier temps une version redondante de cet opérateur, i.e., les éléments obtenus peuvent avoir une intersection non vide.

Définition 7 (Différence de polyèdres)**.** Soient deux polyèdres P_1 et P_2 représentés respectivement par les ensembles de contraintes linéaires C_1 et C_2. La différence de P_1 et P_2 s'écrit alors comme :

$$P_1 \ominus P_2 \triangleq \{(P_1 \cap (\neg c)) | c \in C_2\}$$

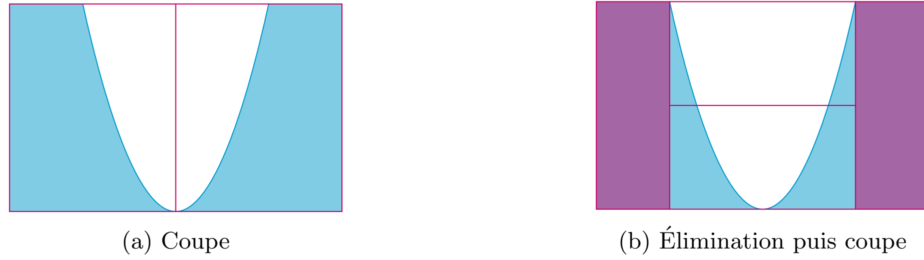
De façon évidente, $\ominus$ est un opérateur de différence pour les polyèdres qui respecte la définititon 6.

Intuition. $P_1 \ominus P_2$ retourne un ensemble qui à chaque contrainte de P_2 associe un élement. Le nombre de contraintes de P_2 étant fini, l'ensemble est donc fini (6.1). De plus, par définition de l'intersection, la propriété (6.2) est respectée car chaque élément retourné est inclu dans P_1. Enfin, (6.3) est aussi respectée : $P_1 \ominus P_2$ peut s'écrire comme $P_1 \cap \overline{P_2}$. P_1 est entièrement couvert par P_2 et $P_1 \ominus P_2$: aucune solution n'est perdue. Toutefois, cet opérateur présente l'inconvénient de retourner une couverture potentiellement redondante de $P_1 \setminus P_2$. Nous proposons alors une amélioration de cette définition pour obtenir une couverture non redondante.

Définition 8 (Différence de polyèdres non redondante)**.** Soient deux polyèdres P_1 et P_2 représentés respectivement par les ensembles de contraintes linéaires C_1 et C_2, avec $C_2 = \{c_1, \ldots, c_{p_2}\}$. On a alors :

$$P_1 \ominus P_2 \triangleq \{(P_1 \cap c_i) \cap (\neg c_j) \mid i, j \in \{1, \ldots, p_2\}, i < j\}$$

Pour des raisons similaires, les propriétés (6.1), (6.2) et (6.3) de la définition (6) sont aussi respectées. Ici, nous nous assurons de plus que pour toute paire d'éléments $(e_1, e_2) \in P_1 \ominus P_2$, $e_1 \cap e_2 = \emptyset$. Ce résultat est garanti par la considération suivante : chaque élément de



(a) Coupe (b) Élimination puis coupe

FIGURE 6 – Comparaison entre la coupe (figure 6a) et l'élimination suivi d'une coupe (figure 6b)

la disjonction est issu de la négation d'une et une seule contrainte et de la conjonction des contraintes précédentes de P_2. Ainsi, pour tout élément e_i de $P_1 \ominus P_2$, tel que e_i est contraint par la négation de la contrainte c_i, on a : e_i a une intersection vide avec les éléments suivant e_j de $P_1 \ominus P_2$ tels que $j > i$ car ceux-ci sont tous contraints par c_i.

La figure 5 montre un exemple d'application de l'opérateur de différence à deux polyèdres. La figure 5a donne les polyèdres P_1 et P_2 de départ, avec P_2 représenté par les contraintes $\{c_1, \ldots, c_5\}$. La figure 5b montre le résultat de l'opérateur de différence redondant, les redondances apparaissent en rose plus foncé sur la figure. La figure 5c montre le résultat de l'opérateur de différence non redondant et permet de constater qu'il n'existe plus de zone plus foncées.

Ici, $P_1 \ominus P_2$ retourne un ensemble de quatre éléments. P_2 est une conjonction de cinq contraintes. La négation d'une d'entre elles est incompatible avec P_1, elle est ignorée.

Nous avons désormais un nouvel opérateur, avec sa définition théorique et opérationnelle. Nous pouvons désormais définir une nouvelle étape dans la résolution utilisant ce nouvel opérateur.

Algorithme 2 Résolution avec élimination

function SOLVE($\mathcal{D}, \mathcal{C}, r$) ▷ $\mathcal{D}$: domaines, $\mathcal{C}$: contraintes
  `sols` $\leftarrow \emptyset$ ▷ solutions trouvées
  `explore` $\leftarrow \emptyset$ ▷ éléments à explorer
  e =`init`($\mathcal{D}$) ▷ initialisation
  push e in `explore`

  while `explore` $\neq \emptyset$ **do**
    $e \leftarrow$ **pop**(`explore`)
    $e \leftarrow$ `filtre`(e,$\mathcal{C}$)
    if $e \neq \emptyset$ **then**
      if $\tau_A(e) \leq r$ **or** `satisfait`(e,$\mathcal{C}$) **then**
        `sols` $\leftarrow$ `sols` $\cup\, e$
      else
        $e_{non-cons} \leftarrow$ `complementaire`(e,$\mathcal{C}$)
        $e_{cons} \leftarrow e \ominus e_{non-cons}$
        for $e_i \in e_{cons}$ **do**
          `sols` $\leftarrow$ `sols` $\cup\, e_i$
        end for
        push $\oplus_A(e \cap e_{non-cons})$ in `explore`
    end if
  end if
  end while
end function

4.2 Nouvelle étape dans la résolution

Calculer $S^\sharp \setminus \overline{S}^\sharp$ peut permettre de réduire l'espace de recheche. En effet, lorsque l'on ne peut plus filtrer les valeurs non-consistantes du domaine des variables, nous proposons une étape d'élimination avant l'étape d'exploration. Plutôt que de diviser arbitrairement un élément abstrait, l'élimination permet d'identifier les parties ne contenant que des solutions. Elle permet d'ajouter une phase de résolution basée sur les contraintes elle-mêmes et délaie ainsi la phase de coupe qui effectue des choix de division plus arbitraires.

Les figures 6a et 6b comparent les résultats d'une coupe d'un élément et d'une élimination suivie d'une coupe pour une contrainte. On remarque que la frontière de coupe ne tient pas compte de la contrainte dans la figure 6a tandis que dans la figure 6b, les boîtes ne contenant que des solutions sont d'abord conservées (les boîtes totalement roses des deux côtés de la parabole). Puis l'opérateur de coupe, coupe la troisième boîte en deux dans la hauteur.

L'algorithme 2 donne le pseudo-code associé à notre méthode de résolution avec la nouvelle étape d'élimination. La différence avec l'algorithme 1 apparait en bleu. La fonction `complementaire` calcule $e_{non-cons}$, une sur-approximation de l'inconsistance. Puis avec l'opérateur de différence les éléments solutions sont conservés. Finalement, la résolution continue dans l'espace de recherche indéterminé ($e \cap e_{non-cons}$).

La figure 7 montre le résultat de cette méthode sur le CSP donné précédemment. On peut voir que sur cet exemple, on obtient pour la même précision : 100 éléments intérieurs et 273 éléments extérieurs en alternant propagation, élimination et exploration, contre

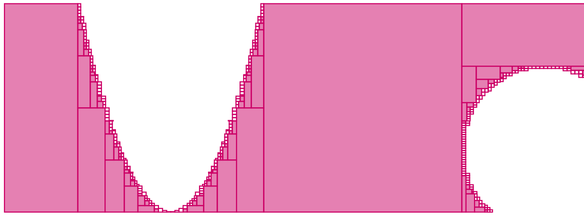


Figure 7 – Résolution avec élimination : 100 éléments intérieurs, 273 éléments frontaliers.

223 éléments intérieurs et 310 éléments extérieurs obtenus en alternant propagation et exploration, soit de deux fois moins d'éléments intérieurs. On analyse dans la section suivante plus finement les performances de notre méthode de résolution.

5 Résultats expérimentaux

Dans cette section, nous expérimentons les performances de la méthode de résolution avec élimination. Nous avons intégré cette technique au sein du solveur AbSolute [1]. Celui-ci implante la méthode présentée initialement dans [17] et la technique d'élimination s'y incorpore particulièrement bien. En effet, celle-ci requiert l'implantation notamment d'une opération de différence ensembliste. Cette opération peut s'avérer coûteuse avec une représentation classique des domaines des variables, tandis que l'utilisation de domaines abstraits permet de réduire son coût.

5.1 Présentation des problèmes

Les problèmes séléctionnés sont des problèmes de satisfaction de contraintes, manipulant des variables continues, qui sont issus du benchmark Coconut [2] et du benchmark MINLP [3]. Les porblèmes *circle*, *mickey* et *exnewton* sont des problèmes classiques d'arithmetique d'intervalle, les problèmes *non-linX* et *pentagon* sont des applications utilisant des systèmes d'équations non linéaires. Le problème *o32* est un problème impliquant des constraintes quadratiques non-convexes.

5.2 Analyse

Le tableau 1 donne les résultats de la méthode de résolution abstraite avec et sans élimination. Pour une précision fixée de $1e^{-3}$ (i.e la taille minimale des éléments retournés), nous nous intéresserons principalement au nombre d'éléments retournés par l'algorithme et au volume couvert correspondant. Les exemples suivants ont été réalisés avec le domaine des intervalles.

Les colonnes 1 et 2 fournissent les informations du problème résolu, à savoir, son nom et, son nombre de variables et de contraintes. Le reste du tableau fournit les informations sur les deux modes de résolution comparés. On a respectivement avec et sans élimination : le temps de résolution en ms (3 et 7, col t), le nombre d'éléments intérieurs (4 et 8, col $\#I$), le nombre d'éléments frontaliers (5 et 9, col $\#E$) et le volume [4] total couvert, (6 et 10, col V). La prise en compte — ou non — d'éléments frontaliers permet d'avoir une résolution respectivement correcte ou complète.

5.2.1 Résultats

On remarque que sur la majorité des problèmes testés, la résolution avec élimination permet de réduire le nombre d'éléments abstraits nécéssaires à la couverture de l'espace des solutions. Aussi, notons que ce gain d'éléments ne se fait pas au détriment du volume couvert, celui-ci étant extremement proche avec les deux modes de résolution. Aussi, les temps de résolution sont un peu plus longs pour la majorité des exemples ce qui s'explique par le fait que l'on manipule plus d'éléments abstraits lors de la résolution : en effet pour chaque élément, la tehnique d'élimination nécéssite le calcul d'un élément complémentaire. Ces bons résultats viennent confirmer l'intuition que couper un élément par rapport aux contraintes qu'il ne satisfait pas est plus intéressant que de le couper arbitrairement sans tenir compte des contraintes. Enfin, notons ici que l'étape d'élimination permet d'effectuer moins d'étapes de propagation et d'exploration.

6 Conclusion et travaux futurs

Nous avons présenté dans cet article un opérateur de différence ainsi que l'étape associée dans la méthode de résolution. Nous nous somme basés sur la méthode de résolution introduite dans [17] et [18], celle-ci est générique et modulaire, basée entièrement sur les domaines abstraits. Nous y avons intégré un mécanisme d'élimination permettant d'améliorer ses résultats en vertus d'un critère quantitatif. Cette technique qui délaie le choix heuristique de l'exploration permet de mieux tirer profit des contraintes d'un problème pour un temps de calcul très peu supérieur. L'opérateur de différence introduit permet dans certains cas de fortement réduire l'espace de recherche et d'affiner les résultats du solveur. Enfin, soulignons que bien qu'implantée dans

1. `https://github.com/mpelleau/AbSolute`
2. `http://www.mat.univie.ac.at/~neum/glopt/coconut/`
3. `http://www.gamsworld.org/minlp/minlplib/minlpstat.htm`

4. Il s'agit du volume d'un hypercube dont le nombre de dimensions est égal au nombre de variables du problème

problem	$\|\mathcal{X}\|, \|\mathcal{C}\|$	sans elimination				avec elimination			
		t	#I	#E	V	t	#I	#E	V
nonlin1	2, 3	**410**	16529	22001	4.551	491	**12474**	18525	4.550
nonlin2	2, 3	**545**	26560	33352	6.088	658	**25510**	30775	6.088
circle	3, 10	128	2136	4903	$5.28 * 10^{-4}$	**125**	**1786**	4906	$5.28 * 10^{-4}$
o32	5, 7	**7922**	1601	103219	$1.4 * 10^{-4}$	10741	848	108889	$1.44 * 10^{-4}$
pentagon	11, 17	**105**	0	11	0.021	200	0	**8**	0.02
booth	2, 2	**342**	8419	10753	1.044	421	8419	10753	1.044
mickey	2, 5	**129**	742	720	2.108	155	**632**	674	2.107
exnewton	2, 3	**158**	7170	7245	0.478	190	**6830**	6834	0.478

TABLE 1 – Comparaison de la méthode de résolution avec et sans élimination

un solveur spécifique à l'utilisation de domaines abstraits, cette technique peut parfaitement être intégrée à un solveur plus classique.

Nous envisageons de déterminer de façon heuristique quand effectuer une étape d'élimination. En effet, lorsque les éléments sont petits, l'étape d'élimination ne conserve que de petites zones solutions et est plus coûteuse qu'elle n'apporte à la résolution. De plus, l'élimnation étant plus efficace pour certaines contraintes, l'heuristique pourrait ainsi déterminer pour quelles contraintes il est plus intéressant de calculer le complémentaire. Par exemple, certaines contraintes non convexes peuvent introduire des composantes non consistantes dans l'espace de recherche. Nous pouvons alors utiliser la technique d'élimination pour *localiser* ces zones et mieux orienter la recherche.

Il serait intéressant de mesurer les performances de cette technique avec d'autres domaines abstraits et d'autres consistance et heuristique de coupe. Finalement, nous aimerions pouvoir essayer cette technique au sein d'autres solveurs.

Références

[1] Heikel Batnini, Claude Michel, and Michel Rueher. Mind the gaps : A new splitting strategy for consistency techniques. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *Lecture Notes in Computer Science*, pages 77–91. Springer-Verlag, 2005.

[2] Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revisiting hull and box consistency. In *Proceedings of the 16th International Conference on Logic Programming*, pages 230–244, 1999.

[3] Julien Bertrane, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, and Xavier Rival. Static analysis and verification of aerospace software by abstract interpretation. In *AIAA Infotech@Aerospace 2010*, Atlanta, Georgia, 20–22 April 2010. American Institute of Aeronautics and Astronautics.

[4] Christian Bessière and Jean-Charles Régin. Mac and combined heuristics : Two reasons to forsake fc (and cbj ?) on hard problems. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, volume 1118 of *Lecture Notes in Computer Science*. Springer, 1996.

[5] Frédéric Boussemart, Fred Hemery, Christophe Lecoutre, and Lakhdar Sais. Boosting systematic search by weighting constraints. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI'2004)*, pages 146–150. IOS Press, 2004.

[6] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4) :251–256, 1979.

[7] Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173 :1079–1100, 2009.

[8] Hélène Collavizza, François Delobel, and Michel Rueher. Extending consistent domains of numeric csp. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 406–413, 1999.

[9] Patrick Cousot and Radhia Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, 1977.

[10] Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96, 1978.

[11] Jérôme Feret. Static analysis of digital filters. In Springer, editor, *European Symposium on Pro-*

gramming (ESOP'04), volume 2986, pages 33–48, 2004.

[12] Eldon Hansen. *Global optimization using interval analysis.* Marcel Dekker, 1992.

[13] Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. In *Proceedings of the 6th International Joint Conference on Artificial intelligence (IJCAI'79)*, pages 356–364. Morgan Kaufmann Publishers Inc., 1979.

[14] Bertrand Jeannet and Antoine Miné. Apron : A library of numerical abstract domains for static analysis. In *Proceedings of the 21th International Conference Computer Aided Verification (CAV 2009)*, 2009.

[15] Antoine Miné. *Domaines numériques abstraits faiblement relationnels*. PhD thesis, École Normale Supérieure, December 2004.

[16] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1) :31–100, 2006.

[17] Marie Pelleau, Antoine Miné, Charlotte Truchet, and Frédéric Benhamou. A constraint solver based on abstract domains. In *Proceedings of the 14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2013)*, 2013.

[18] Marie Pelleau, Charlotte Truchet, and Frédéric Benhamou. The octagon abstract domain for continuous constraints. *Constraints*, 19(3) :309–337, 2014.

[19] Dietmar Ratz. Box-splitting strategies for the interval gauss-seidel step in a global optimization method. *Computing*, 53 :337–354, 1994.

A Procédures annexes de la méthode de résolution

A.1 Procédure d'initialisation

Algorithme 3 Procédure d'initialisation

function INIT($\mathcal{D}$) ▷ $\mathcal{D}$: domaines des variables
  $e' \leftarrow \bot$
  for $i \in \mathcal{D}$ **do**
    $e' \leftarrow e' \cup \alpha(i)$
  end for
  return e'
end function

Cette procédure permet l'initialisation de l'algorithme en sur-approximant les domaines des variables. Il s'agit de l'union des abstractions des valeurs possibles des variables.

A.2 Procédure de filtrage

Algorithme 4 Procédure de filtrage

function FILTRE($e, \mathcal{C}$) ▷ e : un élément abstrait $\mathcal{C}$: contraintes
  $e' \leftarrow e$
  for $c_i \in \mathcal{C}$ **do**
    $e' \leftarrow cons_A(e', c_i)$
  end for
  return e'
end function

Cette procédure permet le filtrage d'un élément par rappot à un ensemble de contraintes. On filtre successivement chaque contrainte de l'élément à l'aide de l'opérateur de consistance. Différentes consistances sont possibles. AbSolute implémente l'algorithme `Bottom-Up, Top-down` de [15, §2.4.4] développé indépendamment en PPC sous le nom de `HC4-revise` de [2].

A.3 Test de satisfaction

Algorithme 5 Test de satisfaction de contraintes

function SATISFAIT($e, \mathcal{C}$) ▷ e : un élément abstrait $\mathcal{C}$: contraintes
  for $c_i \in \mathcal{C}$ **do**
    if $cons_A(e, \neg c_i) \neq \bot$ **then**
      return false
    end if
  end for
  return true
end function

Cette procédure prend un élément abstrait et une liste de contraintes et retourne vrai si et seulement si l'élément abstrait satisfait toutes les contraintes. On considère qu'un élément satisfait une contrainte si il possède une intersection nulle avec la négation des contraintes. On pourrait être ici plus précis/efficace en utilisant un test de satisfaction spécifique par domaine abstrait, mais cette méthode, plus générique facilite l'implémentation de nouveaux domaines.

A.4 Complémentaire

Algorithme 6 Complémentaire

function COMPLEMENTAIRE($e, \mathcal{C}$) ▷ e : un élément abstrait $\mathcal{C}$: contraintes
  $e' \leftarrow \emptyset$
  for $c_i \in \mathcal{C}$ **do**
    $e' \leftarrow e' \cup cons_A(e, \neg c_i)$
  end for
  return e'
end function

Cette procédure prend un élément abstrait et un ensemble de contrainte $\mathcal{C}$. Elle retourne un élément e' inclut dans e correspondant à un espace complémentaire de celui definit par les contraintes de $\mathcal{C}$.