Contraintes sur les réels et contraintes sur les flottants: contributions.

Claude Michel

Habilitation à Diriger des Recherches

Présentée à

L'Université de Nice-Sophia Antipolis
École doctorale STIC
Spécialité : Informatique

par

Claude Michel

Contraintes sur les réels et contraintes sur les flottants : contributions.

Soutenue publiquement le jeudi 7 juillet 2016
après avis des rapporteurs

M. **Éric Goubault**	Professeur, École Polytechnique
M. **Pascal Van Hentenryck**	Professeur, Université du Michigan
M. **Andréas Podelski**	Professeur, Université de Freiburg

devant la commission d'examen composée de

M. **Gilles Bernot**	Professeur, Université de Nice-Sophia Antipolis
M. **Roberto di Cosmo**	Professeur, Université Paris-Diderot
M. **Éric Goubault**	Professeur, École Polytechnique
M. **Luc Jaulin**	Professeur, ENSTA Bretagne
M. **Jean-Michel Muller**	Directeur de Recherches, CNRS, ENS Lyon
M. **Andréas Podelski**	Professeur, Université de Freiburg
M. **Michel Rueher**	Professeur, Université de Nice-Sophia Antipolis

Table des matières

Synthèse des Travaux

1 Introduction

Ce document fait la synthèse des travaux de recherche que j'ai effectués après ma thèse [72], thèse qui traitait de la réflexivité. Ces recherches portent principalement sur la programmation par contraintes, avec deux thèmes de prédilection, les contraintes sur le continu et les contraintes sur les flottants.

La programmation par contraintes (PPC) est un paradigme caractérisé par une séparation claire entre la modélisation d'un problème et sa résolution. Idéalement, l'utilisateur se contente de décrire le problème dont la résolution est confiée à la machine. La description du problème repose sur un triplet (variables, domaines, contraintes) où les domaines des variables définissent l'ensemble des valeurs qu'elles peuvent prendre et les contraintes sont des relations entre ces variables qui en restreignent les valeurs. Cette approche souple permet de décrire une grande variété de problèmes allant de problèmes linéaires sur les entiers à des problèmes non linéaires sur les réels et de résoudre des problèmes d'ordonnancement, de robotique ou encore, de vérification de programmes.
La séparation entre modèle et résolution rejette les principales difficultés vers les méthodes de résolution. Naturellement, mes recherches ont porté essentiellement sur ces techniques de résolution, que ce soit sur la résolution de problèmes non linéaires sur le continu, où les domaines des variables sont des sous-ensembles de réels, que sur la résolution de problèmes de contraintes sur les flottants, un nouveau type de contraintes dédiées à l'analyse de programmes où les domaines des variables sont des sous-ensembles de flottants.

Contraintes sur le continu. La résolution de contraintes sur le continu s'appuie le plus souvent sur une relaxation de la consistance d'arc aux bornes du domaine des variables. Cette relaxation a donné naissance à la $2B$-consistance [66] et la Box-consistance [10], deux consistances locales qui calculent les réductions des domaines en utilisant une seule contrainte à la fois. De la localité de ce raisonnement naît une limitation sur les réductions possibles des domaines. Lorsque nous nous sommes intéressés à la résolution

de problèmes quadratiques, nous avons observé que de nombreux problèmes offraient des opportunités de réductions substantielles à une échelle plus globale. Nous avons donc proposé un filtrage global pour des contraintes sur le continu en utilisant les techniques de relaxations introduites par Sahinidis et al. [96, 97, 101]. Plus précisément, en plongeant ces techniques, dites RLT, pour techniques de relaxations et linéarisations, dans un solveur de contraintes, nous avons obtenu un filtrage plus global des contraintes quadratiques [65, 60]. Concrètement, une minimisation et une maximisation appliquées à chaque variable du problème initial permettent d'en réduire le domaine en tenant compte de l'ensemble du problème linéarisé.

Les résultats encourageants auxquels nous sommes parvenus nous ont conduits à généraliser cette approche au polynomial [57, 64, 59]. Le solveur étendu à ces relaxations s'est révélé particulièrement efficace pour résoudre des problèmes de robotique qui présentent des structures favorables à ce type de filtrage global. Notons cependant que lorsque les filtrages locaux capturent correctement le problème, les nombreux appels au simplexe peuvent se révéler plus coûteux que le bénéfice apporté.

Pour garantir de bonnes performances, ce filtrage global s'appuie sur une implémentation efficace du simplexe, i.e., une implémentation sur les flottants. L'utilisation de calculs sur les flottants sans tenir compte de leurs limites conduit à la perte de solutions, en particulier, lorsque le minimum calculé se trouve au-dessus du minimum réel. Une autre source potentielle de pertes de solutions réside dans le calcul des linéarisations : le calcul des coefficients de ces linéarisations avec des opérations sur les flottants soumises à des arrondis a pour effet d'en modifier la position. Afin d'obtenir un solveur correct, i.e., qui ne perd pas de solution sur les réels, nous avons proposé des calculs rigoureux des coefficients des linéarisations et utilisé un calcul correct du minimum global [75, 58, 64, 60].

Afin d'améliorer le processus de recherche des solutions, nous nous sommes basés sur une idée simple, l'utilisation de "trous", i.e., d'espaces contigus de non-solutions présents à l'intérieur des domaines des variables, comme critère de choix et de coupes. Ces espaces non solutions restent le plus souvent ignorés afin d'éviter la multiplication des intervalles liés au domaine d'une variable et l'explosion combinatoire qui s'ensuivrait. Toutefois, en modifiant légèrement les filtrages locaux, il est possible de les retrouver et de les exploiter dans le cadre de la recherche sans générer les inconvénients qui apparaissent dans le cadre du filtrage [3, 2]. Ces travaux sont à la base de la thèse d'Heikel Batnini [1] que j'ai co-encadré.

J'ai ensuite travaillé sur la résolution de problèmes d'optimisation globale. Pour résoudre ces problèmes, nous nous sommes appuyés sur un ensemble de techniques correctes de linéarisation d'un problème non linéaire pour calculer une borne inférieure de la fonction objectif. Nous les avons combinées avec des filtrages classiques pour réduire la taille des domaines

des variables. Une recherche locale sécurisée par des tests d'existence de solutions permet de calculer une borne supérieure de la fonction objectif. Notre algorithme d'évaluation et séparation, ou "branch and bound", combine donc relaxations linéaires correctes et techniques de filtrage classiques pour résoudre efficacement des problèmes d'optimisation globale [58, 61].

Dans ce contexte, nous nous sommes aussi intéressés à l'implémentation rigoureuse de la réduction basée sur l'optimalité [96]. Cette technique cherche à exploiter les bornes connues de la fonction objectif pour réduire la taille des domaines de ses variables. Des tentatives de corrections de la méthode par une approche mathématique rigoureuse se sont révélées incomplètes et inefficaces [55, 56]. Nous avons donc proposé une approche rigoureuse où la programmation par contraintes tente de valider, par un processus de réfutation, les réductions calculées et, lorsqu'elle échoue, les utilise comme des coupes des domaines des variables. Cette approche, simple et naturelle dans le contexte de la programmation par contraintes, s'est avérée plus efficace que l'approche rigoureuse (mais incomplète) de Kearfott [62, 63].

Pour améliorer l'efficacité du calcul de la borne supérieure de la fonction objectif, j'ai proposé d'utiliser une information mise à disposition par le calcul de la borne inférieure, à savoir la solution de l'optimum global du problème linéaire. La principale difficulté est que cette solution se trouve en dehors de la zone faisable alors que la borne supérieure nécessite un point faisable, i.e., une solution du problème. Nous avons donc introduit un algorithme basé sur la méthode de Newton qui permet de calculer une pseudo-solution aussi proche que nécessaire de l'ensemble faisable à partir de la solution de la relaxation. Cette pseudo-solution sert ensuite à la construction par "inflation" d'une boîte contenant des solutions faisables [37].

Toujours sur le continu, j'ai travaillé à la résolution de contraintes universellement quantifiées, plus précisément sur une sous-classe où les quantificateurs existentiels précèdent les quantificateurs universels, une classe qui couvre, entre autres, des problèmes de conception et de robustesse. La résolution repose sur une approche par contraintes basée sur une observation essentielle : il est possible de remplacer les paramètres, quantifiés universellement, par une instance de leur domaine. La méthode de résolution proposée [38, 39] se montre particulièrement efficace et s'intègre naturellement dans un solveur de contraintes.

Contraintes sur les flottants. En parallèle de mes travaux sur le continu, j'ai contribué à l'introduction des contraintes sur les flottants.

Les contraintes sur les flottants constituent une nouvelle catégorie de contraintes née de la nécessité de traiter, en analyse et vérification de programmes, des expressions qui dénotent des calculs sur les flottants. Alors que de nombreux travaux avaient montré l'intérêt de la programmation par

contrainte dans la vérification de programme [26, 41, 87], à notre connaissance, aucun n'avait encore pris en compte la nécessité de traiter des calculs sur les flottants. La nature particulière des opérations sur les flottants n'est pas correctement capturée par les contraintes classiques, que ce soient les contraintes sur les rationnels, ou les contraintes sur le continu, aucune d'entre elles n'étant conservatives des solutions sur les flottants. Nous avons donc introduit un cadre pour la résolution des contraintes sur les flottants, cadre où un processus dit de "shaving" qui s'appuie sur l'arithmétique des intervalles est utilisé pour réduire la taille des domaines sans perte de solution sur les flottants [77].

Bien que correct, ce premier filtrage sur les flottants manque d'efficacité : les arrondis extérieurs de l'arithmétique des intervalles sont surconservatifs et le procédé de "shaving" utilisé nécessite des calculs longs. J'ai donc travaillé à l'amélioration de l'efficacité du filtrage sur les flottants en proposant une méthode plus proche d'une consistance d'intervalles. La principale difficulté était de permettre de réduire les domaines des variables d'entrée d'une contrainte élémentaire sur les flottants, i.e., de calculer les réductions du domaine de x pour une contrainte de type $z = x + y$. À cette fin, j'ai donc proposé des projections inverses conservatives des solutions sur les flottants [73], travaux que j'ai ensuite étendus, en collaboration avec Bernard Botella et Arnaud Gotlieb [13].

Alors que ce second filtrage est nettement plus efficace, des problèmes de convergences lentes sont apparus au sein même des contraintes ternaires, en particulier dans le cas de l'addition et de la soustraction. C'est avec Bruno Marre que nous avons introduit une propriété de la soustraction sur les flottants qui permet de remédier à ces cas de convergences lentes en calculant des réductions sur les domaines des opérandes à partir de la seule connaissance du domaine de sortie de l'addition ou de la soustraction [70].

L'arithmétique des flottants souffre de nombreuses limitations (e.g., perte de la distributivité, de l'associativité, etc.) et nécessite le strict respect de l'ordre des opérations afin de reproduire un résultat identique. D'un autre côté, les réels bénéficient de beaucoup plus de flexibilité et de nombreux algorithmes de résolutions conservatifs des solutions, y compris lors du calcul sur machine ; au moins pour ceux qui ont pris en compte la problématique des flottants dans leur implémentation. Une idée séduisante est donc de plonger les contraintes des flottants dans les réels, i.e., de les relaxer dans $\mathcal{R}$ en proposant des approximations conservatives des solutions sur les flottants [6, 7, 74] et, ainsi, de bénéficier des approches sur les réels. Cette idée est à la base de la thèse de Mohammed Said Belaid [5] que j'ai co-encadré.

L'interprétation abstraite est une technique d'analyse de programmes qui a montré ses capacités à passer à l'échelle et qui offre des techniques intéressantes de traitement des boucles, deux caractéristiques qui sont des points faibles de la programmation par contraintes. Cette dernière permet cependant des approximations plus précises et a la capacité de produire des contre-

exemples, i.e., des jeux de valeurs des variables d'entrée pour lesquelles un phénomène non désirable se produit. Afin de remédier aux défauts respectifs de ces deux approches, nous avons proposé un schéma de collaboration entre ces deux techniques qui tente d'en combiner les avantages [88, 89].

J'ai aussi travaillé sur la génération de cas de tests d'intervalles suspects [20], i.e., des intervalles pour lesquels le programme pourrait adopter un comportement non souhaitable en suivant un chemin différent sur les réels et sur les flottants. La détection de ces comportements est cruciale dans le cadre d'applications critiques pour lesquelles des divergences de chemins pourraient conduire à des comportements catastrophiques.

Autre travaux. De manière plus annexe, j'ai aussi effectué des recherches sur la résolution de problèmes de cohérences d'une installation Linux lors de l'ajout, de la mise à jour ou du retrait de paquets [76, 22]. Dans le cadre du projet européen Mancoosi, j'ai, en particulier, réalisé *mccs* (multi criteria CUDF solver), un solveur qui recherche la meilleure solution selon des critères multiples au problème précédent. Cet outil est disponible dans la distribution Debian. Ces recherches portant sur un sujet trop éloigné de l'axe thématique de cette synthèse, elles ne sont pas décrites plus en détail dans ce document.

Nombre de ces recherches ont été effectuées grâce à des contrats de recherches nationaux et internationaux. J'ai participé activement à la construction et à la réalisation de la plupart d'entre eux. On compte parmi ces projets, le RNTL Inka, l'ACI V3F, le RNTL Danocops, l'ANR Cavern, le projet européen FP7 Mancoosi, l'ANR Aeolus, l'OSI ISI Pajero et l'ANR Coverif qui a débuté en 2015. Ces projets qui m'ont permis de rencontrer différents acteurs de la recherche ont été le socle de collaborations fructueuses et un environnement d'échanges très intéressants.

Notations. Dans la suite de ce document, $\mathcal{R}$ dénote l'ensemble des réels et $\mathcal{F}$ un ensemble de nombres à virgule flottante. Pour ce dernier, sauf précision contraire, il s'agit de l'un des ensembles de flottants en base 2 tel que décrit dans la norme IEEE 2008. x est utilisé indifféremment pour désigner une variable ou un vecteur de variables, l'ambiguïté étant levée par le texte. Lorsque x est un vecteur, ses différents éléments sont distingués par un indice (e.g., x_2 est la seconde composante du vecteur x). Les domaines de valeurs des variables sont des intervalles de réels ou de flottants. Ils sont dénotés par le même nom que la variable, mais en gras. Plus précisément, $\mathbf{x} = \{x \in \mathcal{R}, \underline{x} \leq x \leq \overline{x}\}$ est un intervalle fermé de réels dont la borne inférieure, $\underline{x}$, et la borne supérieure, $\overline{x}$, délimitent les valeurs possibles. De manière plus concise, on note $\mathbf{x} = [\underline{x}, \overline{x}]$. $]\underline{x}, \overline{x}[= \{x \in \mathcal{R}, \underline{x} < x < \overline{x}\}$ dénote un intervalle ouvert de réels. De même que x peut représenter un vecteur, $\mathbf{x}$ peut dénoter un vecteur d'intervalles. L'extension naturelle aux intervalles d'une fonction

f est dénotée par $\mathbf{f}$ (i.e., le nom de la fonction mis en gras). Les majuscules représentent des matrices, que ce soient des matrices de scalaires comme A, ou encore d'intervalles telle que $\mathbf{A}$. Soit $x \in \mathcal{F}$, un flottant, x^- est le prédécesseur de x et x^+ est le successeur de x.

Une table des principales notations utilisées est disponible en page 46.

2 Contraintes sur le continu

2.1 Introduction

Cette section présente mes principales contributions liées aux contraintes sur le continu. Ces contraintes se caractérisent par un triplet (variables, domaines, contraintes) où les domaines des variables sont des sous-ensembles des réels représentés par des intervalles de réels bornés par des flottants afin d'en permettre la représentation et la manipulation en machine. En d'autres termes, soit la variable x, son domaine $\mathbf{x} = [\underline{x}, \overline{x}]$ est défini par l'ensemble $\{\underline{x} \leq x \leq \overline{x}, x \in \mathcal{R}, \underline{x} \in \mathcal{F}, \overline{x} \in \mathcal{F}\}$ où $\underline{x}$, et $\overline{x}$ sont, respectivement, les bornes inférieures et supérieures de l'intervalle $\mathbf{x}$. Les problèmes traités ici relèvent du non linéaire sans, a priori, de limitations quant à la complexité des expressions utilisées. Cependant, les techniques sous-jacentes couramment utilisées comme le Newton multivarié par intervalles ou certaines preuves d'existence de solutions reposent sur des fonctions continûment différentiables. L'axe principal de ces recherches est l'utilisation correcte de relaxations linéaires pour résoudre des problèmes de contraintes, ou pour déterminer l'optimum global d'un problème sous contraintes. Des résultats sur le traitement de contraintes universellement quantifiées sont aussi présentés en fin de section.

2.2 Filtrage global de contraintes sur le continu

2.2.1 Introduction

La résolution de contraintes sur le continu suit habituellement un schéma en deux étapes :

- une étape de filtrage des domaines dont l'objectif est d'en réduire autant que possible la taille ;
- une recherche de solutions où, en général, une des variables est choisie et son domaine divisé en plusieurs sous-domaines afin de générer des sous-problèmes de plus petite taille.

Ces deux étapes sont appliquées alternativement aux sous-problèmes générés jusqu'à l'obtention d'une solution, ou à l'absence de nouveau sous-problème, i.e., l'absence de solution. Les filtrages utilisés s'appuient sur des consistances locales, telles que la $2B$-consistance [66] ou la Box-consistance [10], qui peuvent être vues comme une relaxation de la consistance d'arc aux bornes des domaines des variables. À quelques nuances près, liées aux problèmes de représentation en machine des réels, ces consistances sont véri-

fiées lorsque les bornes des domaines sont les solutions, la plus à gauche et la plus à droite, d'une contrainte donnée. Ce raisonnement à l'échelle d'une contrainte, dont les effets sont répercutés sur les autres contraintes par un mécanisme de propagation, limite les réductions obtenues sur les domaines des variables. Lorsqu'un raisonnement plus global est nécessaire, des consistances plus fortes sont utilisées comme les kB-consistances [66] et la *Bound*-consistance [91]. Mais ces consistances fortes qui reposent sur des consistances moins fortes restent assez lourdes à calculer. Notons aussi que le Newton multivarié souvent utilisé dans l'implémentation de la *Box*-consistance lui donne un certain degré de globalité.
Les approches décrites ici mettent en œuvre des techniques de linéarisation qui sont le support d'un raisonnement plus global sur le problème à résoudre. C'est la première fois, à notre connaissance, que ces techniques ont été utilisées à une telle échelle au sein d'un solveur de contraintes. C'est aussi la première fois qu'elles le sont de manière correcte et rigoureuse pour offrir des garanties de conservations de l'ensemble des solutions.
Nous décrivons aussi dans cette section des travaux sur l'utilisation des espaces non solutions identifiés par les filtrages comme stratégie de recherche des solutions.

2.2.2 Filtrage global de contraintes quadratiques

De nombreux problèmes s'expriment sous forme quadratique que ce soient des contraintes géométriques de distances telles que celles inhérentes à certains problèmes de robotique, ou encore de nombreux problèmes de physique. Or un simple système d'équations et d'inéquations contenant des termes quadratiques n'est pas facile à résoudre, en particulier sur les réels. Nous nous sommes donc intéressés à la résolution de ces systèmes en proposant une approche nouvelle dans le monde des systèmes de contraintes sur le continu.

L'approche prônée ici est celle d'un filtrage fort, i.e., capable d'obtenir des réductions significatives des domaines des variables au prix d'un calcul assez important. Il s'appuie sur une relaxation linéaire du problème initial qui s'effectue en deux temps. Lors de la première étape, le sous-problème linéaire est capturé en substituant à chaque terme non linéaire une variable dont le domaine est estimé à l'aide d'un simple calcul d'intervalles. Si le problème est linéaire, alors aucune information n'est perdue lors de cette étape. La seconde étape construit des relaxations linéaires fines des sous-termes non linéaires, telles que le produit de deux variables ou le carré d'une variable. Le choix des relaxations utilisées est crucial : une relaxation trop fine, comme l'enveloppe convexe du carré d'une variable, risque d'engendrer un surcoût de calcul trop important et une relaxation trop grossière risque de ne pas permettre de réduire les bornes des variables. Un équilibre entre précision des relaxations et efficacité du calcul est donc nécessaire. Cela nous a, entre autres, conduits à réduire le nombre de linéarisations générées pour un carré

à trois, i.e., les tangentes en $\underline{x}$ et $\overline{x}$ ainsi que la droite liant $(\underline{x}, \underline{x}^2)$ à $(\overline{x}, \overline{x}^2)$. Naturellement, le caractère positif du domaine du calcul est lui aussi pris en compte.

L'algorithme de filtrage [65, 60] repose sur une minimisation et une maximisation de chacune des variables du problème. Chaque modification de domaine est intersectée avec le domaine initial afin de garantir que l'algorithme converge vers un point fixe. Le coût de calcul est donc assez important (deux appels de simplexe par variable), mais les réductions possibles sont conséquentes. Des comparaisons de résolution de systèmes quadratiques avec une $2B$-consistance, une $3B$-consistance, ou encore une Box-consistance montrent des gains significatifs en temps de calcul pour ce filtrage. Notons que nous ne parlons pas ici de contrainte globale : en effet, aucune construction linguistique particulière n'est ajoutée à l'ensemble des contraintes du problème. Les relaxations sont construites automatiquement à partir du CSP sans que l'utilisateur n'ait à le spécifier. Ce n'est que le filtrage qui capture un sous-problème, assez large, du problème initial. Toutes les non-linéarités n'étant pas capturées à ce niveau, ces relaxations sont toujours utilisées en conjonction avec une consistance locale classique. Cette dernière effectue des réductions complémentaires.

2.2.3 Extension au polynomial

L'approche du quadratique proposée dans [65] est assez générale pour traiter des formes non linéaires plus complexes telles que celles rencontrées dans les polynômes. Il suffit pour cela de l'étendre aux produits multiples et aux puissances supérieures à 2.

De manière générale, les différentes non-linéarités d'un polynôme peuvent être gérées par un processus appelé *quadrification* [100]. Il consiste en une décomposition des termes de degré supérieur à 2 en combinaison de termes de degré 2. Par exemple, le produit multiple $x_1x_2x_3$ est équivalent à l'ensemble des produits $x_{12}x_3$, x_1x_{23} et x_2x_{13} où x_{ij} est la variable associée au produit x_ix_j. Une telle décomposition ne génère que des termes bilinéaires dont les relaxations sont bien connues [71]. Pour limiter l'introduction de nouvelles variables et la multiplication des relaxations, une quadrification partielle est opérée. Dans l'exemple ci-dessus, elle consiste à ne prendre en compte que la première quadrification. Ainsi, le nombre de variables générées est réduit à $(n-1)$ et celui des linéarisations à $4(n-1)$. Une technique de reformulation/linéarisation directe des polynômes, sans passer par un processus de quadrification, a été proposée par Sherali et Tuncbilek [99]. Bien que plus précise, elle génère beaucoup plus de variables et linéarisations qu'une quadrification partielle.

Une puissance de la forme x^n est approximée par $n+1$ inéquations suivant une procédure proposée par Sherali et Tuncbilek [99] et appelée "bound-factor product RLT constraints". Ces inéquations définies par l'en-

semble $\{[(x - \underline{x})^i(\overline{x} - x)^{(n-i)}], i \in \{1..n\}\}$ génèrent de multiples relations entre les différentes puissances de x, relations d'un grand intérêt lorsque plusieurs puissances d'un même terme apparaissent au sein du problème. D'autres relaxations des puissances ont été proposées par [97, 67], mais elles ne construisent pas de relations aussi riches entre les différentes puissances d'un même terme. Une combinaison de ces relaxations plus fines avec une génération des relations entre les différentes puissances d'un même terme apparaissant au sein du problème mériterait une étude comparative.

Les expérimentations [57, 64, 59] que nous avons menées avec le solveur que nous avons développé ont montré un bon comportement de cette approche sur des problèmes non triviaux, en particulier sur le problème de la plateforme de Gough-Stewart [28], un type de robot parallèle constitué de six actionneurs.

2.2.4 Traitement correct des relaxations

L'utilisation de relaxations et d'un simplexe soulève de réels problèmes en termes de conservation de l'ensemble des solutions. Les relaxations résultent de calculs de coefficients, calculs effectués sur les flottants. Les arrondis inhérents à ces calculs modifient ces relaxations et peuvent conduire à des pertes de solutions. Par ailleurs, les implémentations efficaces de l'algorithme du simplexe font largement appel à des calculs sur les flottants. L'objectif ainsi calculé peut lui aussi être sensiblement différent de sa valeur mathématique sur les réels. Un tel comportement conduit naturellement à se poser la question de la préservation de l'ensemble des solutions, question particulièrement sensible dans le contexte des CSP sur les réels où la résolution d'un problème s'appuie sur une stratégie de bissection des domaines couplée avec le filtrage.

Nous nous sommes donc intéressés à la préservation des solutions lors d'un filtrage global basé sur des relaxations linéaires avec un traitement adapté des relaxations afin d'obtenir une représentation conservative de l'ensemble des solutions du problème initial et l'utilisation d'une procédure calculant un minimum garanti de la fonction objectif [75, 58, 64, 60].

Pour illustrer notre propos, observons ce qui se passe sur un petit exemple illustratif :

$$\begin{aligned} & y - 2 * x \geq 0 \\ & y + 2 * x \geq 0 \\ & -y \geq k \\ & \text{with } x \text{ and } y \in [-2.0, 2.0] \end{aligned}$$

Lorsque $k = 0$, même un simplexe sur les flottants comme CPLEX trouve la solution $x = 0$ et $y = 0$. Supposons que k résulte d'un calcul sur les flottants dont les arrondis sur les opérations produisent non plus 0, mais son successeur, i.e., $k = 0^+$. Alors, plus aucune solution n'est trouvée. De fait, le changement de valeur de k a transformé un problème satisfiable en problème

insatisfiable.

Afin d'obtenir un système linéaire conservatif de l'ensemble des solutions sur les réels, nous avons proposé un calcul correct des coefficients des linéarisations utilisées. Une première approche [75] a consisté à déterminer les directions des arrondis en fonction des différentes valeurs de x pour chacun des coefficients des linéarisations utilisées dans le quadratique de manière à garantir la conservation des solutions. Avec l'introduction de différentes linéarisations, nous avons proposé une méthode plus générale basée sur le calcul par intervalles [58] pour une forme linéaire $\sum_i a_i x_i + b \geq 0$ où b et les a_i sont calculés sur les flottants.

Notons que Hongthong et Kearfott [50] se sont basés sur ces travaux pour proposer des corrections pour d'autres linéarisations. De même, Borradaile et Van Hentenryck [12] ont généralisé notre approche et proposé une formulation mathématique plus rigoureuse de ce problème.

La correction du modèle n'est cependant pas suffisante pour une utilisation correcte d'un simplexe efficace, c'est-à-dire, sur les flottants. Les calculs qu'effectuent les algorithmes standard du simplexe sur les flottants n'étant pas rigoureux, le minimum global qu'ils produisent est parfois au-dessus du vrai minimum global. Dans le contexte de l'utilisation de linéarisations pour réduire les bornes des domaines des variables, cela correspond à une perte de solutions. Dans [83], Neumaier et Shcherbina décrivent une méthode peu coûteuse qui permet d'obtenir un minimum global rigoureux. C'est sur cette méthode que nous nous sommes appuyés afin de garantir la conservation des solutions.

La correction des coefficients des linéarisations du problème alliée à celle du minimum global calculé par le simplexe permet d'obtenir un algorithme de résolution rigoureux qui s'intègre bien dans un solveur de contraintes.

Plus de détails sur ces travaux se trouvent dans [75, 58, 64, 60].

2.2.5 Espaces non solutions et recherche de solutions

Les méthodes classiques de résolution de CSP sur le continu se basent sur un algorithme qui alterne une étape de bissection avec une étape de filtrage. La plupart des solveurs par intervalles utilisent des consistances locales, telles que la $2B$-consistance [66, 8] ou la Box-consistance [10], ou des consistances partielles comme la kB-consistance [66] ou la $Bound$-consistance [91]. Les algorithmes de filtrage associés à ces consistances calculent toutes les données nécessaires à l'identification des espaces de non-solution strictement contenus à l'intérieur des domaines. Toutefois, ces informations n'y sont utilisées que pour calculer le plus petit intervalle contenant toutes les solutions.

Nous avons introduit une nouvelle stratégie de recherche [2, 3], nommée MindTheGaps, qui exploite les espaces non solutions identifiés par les algorithmes de filtrage. Ces espaces non solutions, collectés pour un coût négligeable, servent au choix de la variable à couper ainsi qu'à celui de la

coupe au sein du domaine de cette variable. Effectuer une coupe qui supprime ces espaces non solutions diminue l'espace de recherche de manière significative. Cela permet aussi l'élimination de certaines solutions redondantes et aide l'algorithme de recherche à isoler les solutions. Lorsqu'aucun espace non solution n'est identifié, une stratégie classique est adoptée.

Une stratégie similaire avait été suggérée par Hansen [46, 48] pour le Newton par intervalles. La stratégie de recherche exploite les espaces non solutions identifiés par la méthode de Gauss-Seidel. Elle a été mise en œuvre dans le cadre de l'optimisation globale par Ratz [95]. Nous avons étendu ces stratégies aux consistances classiques, i.e., la $2B$-consistance, la *Box*-consistance et la *Bound*-consistance.

Afin de ne pas générer un nombre trop important d'espaces non solutions, ceux produits par les fonctions trigonométriques ont été ignorés. De fait, les espaces non solutions exploités sont ceux produits par les puissances et les divisions, opérations qui n'en produisent qu'un seul à la fois. Par exemple, pour une division x/y en arithmétique étendue des intervalles [49], lorsque $0 < \underline{x}$ et $\underline{y} < 0 < \overline{y}$, le résultat est donné par l'union de $[-\infty, \underline{x}/\underline{y}]$ et $[\underline{x}/\overline{y}, +\infty]$; $]\underline{x}/\underline{y}, \underline{x}/\overline{y}[$ est donc un espace non solution.

Alors que pour la $2B$-consistance, les espaces non solutions sont produits par les fonctions de projections, la *Box*-consistance peut en produire grâce au Newton univarié qu'elle utilise pour renforcer la consistance des bornes de ses domaines. En effet, l'opérateur du Newton univarié, $N(\mathbf{f}, \mathbf{x}, x') = x' - \mathbf{f}(x')/\mathbf{f}'(\mathbf{x})$, avec $x' \in \mathbf{x}$, utilise une division susceptible de produire des espaces non solutions.

Le principe de réfutation à la base des kB et *Bound*-consistances est lui aussi générateur d'espaces non solutions : pour renforcer la consistance de la borne supérieure du domaine de x, ces consistances tentent de prouver que pour $x' \in \mathbf{x}$, $[x', \overline{x}]$ est inconsistant. Lors de ce processus, si la borne inférieure de $[x', \overline{x}]$ est réduite à $x'' > x'$, alors $[x', x'']$ ne contient pas de solution et est donc, par définition, un espace non solution. En modifiant légèrement les algorithmes de base, il est donc possible de collecter des espaces non solutions.

Une légère modification de la recherche de solutions afin qu'elle puisse exploiter ces informations s'est traduite par une amélioration notable des temps de résolutions. Parmi les différentes stratégies explorées, le choix de la variable avec l'espace non solution le plus important suivi d'une coupe sur ce même espace non solution a donné les résultats les plus constants. Cette idée simple, exploiter les espaces non solutions naturellement produits par les filtrages pour améliorer la recherche de solutions, et peu coûteuse, s'est révélée plus payante qu'il n'aurait paru au premier abord.

Ce travail a été effectué dans le cadre de la thèse d'Heikel Batnini [1] que j'ai co-encadré.

2.2.6 Conclusion

Le filtrage global rigoureux décrit ici a permis d'améliorer notablement le temps de résolution de problèmes sur le continu. Il se caractérise par un schéma clair d'interactions entre relaxations linéaires/simplexe et filtrages classiques sur le continu. La capture du sous-problème linéaire et la construction des relaxations linéaires s'effectuent de manière transparente pour l'utilisateur final. Combiné avec une prise en compte rigoureuse des calculs sur les flottants, ce filtrage global offre la garantie que toutes les solutions sont préservées.

Pour nuancer ces propos, notons néanmmoins que ce filtrage repose sur $2n$ appels au simplexe pour n variables. Un appel au simplexe est une opération très coûteuse qui impacte l'efficacité du filtrage, et lorsque le problème est bien traité par les filtrages classiques le coût de ces appels grève les bénéfices attendus. Dans le cadre de la programmation par contraintes, un tel filtrage ne se montre intéressant que lorsque le point de vue global qu'apporte le simplexe peut bénéficier à la résolution du problème, en particulier, si le problème comporte des occurrences multiples de variables, que ce soit au sein d'une même contrainte ou dans différentes contraintes. Une analyse de la structure du problème pourrait permettre de déterminer quand l'appel à ce filtrage présente un réel bénéfice.

2.3 Optimisation globale

2.3.1 Introduction

Cette section présente nos travaux sur l'optimisation globale de problèmes non linéaires sous contraintes sur le continu. Ces problèmes, particulièrement difficiles à résoudre, consistent à trouver une solution qui minimise ou maximise une fonction objectif dans un espace faisable caractérisé par les domaines des variables et un ensemble de contraintes. Notre approche se caractérise par le mariage de deux tendances jusqu'alors antagonistes, les approches rigoureuses basées sur l'arithmétique des intervalles dont l'objectif est la correction des calculs, et les approches à base de relaxations linéaires prônées par les adeptes de l'efficacité.

Nous montrons aussi comment des techniques complémentaires qui permettent d'améliorer l'efficacité du solveur peuvent être implémentées rigoureusement en utilisant la programmation par contrainte et comment la recherche d'une borne inférieure de la fonction objectif peut bénéficier des informations calculées par les relaxations linéaires.

2.3.2 Relaxations linéaires et optimisation globale

Parmi les différentes approches développées pour résoudre des problèmes d'optimisation, deux grandes tendances se dégagent. La première, et proba-

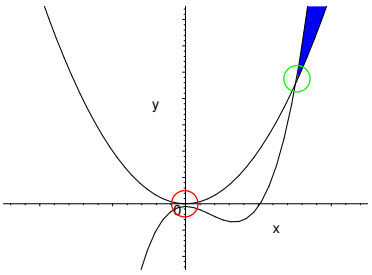

FIGURE 1 – Représentation géométrique du problème 1

blement celle qui a le plus de succès, se concentre sur la rapidité de résolution du problème. Des relaxations linéaires et des méthodes de recherche locale sont utilisées pour accélérer la convergence vers l'optimum global. Le représentant le plus connu de cette approche est Baron [101, 97], l'optimiseur global de Sahinidis. Toutefois, bien que rapides et *complètes*, ces méthodes ne sont pas *rigoureuses* : le résultat de ces algorithmes peut être une sur estimation, ou pire, une sous-estimation de l'optimum global. La seconde tendance s'appuie principalement sur le calcul par intervalles pour offrir des bornes *rigoureuses* de l'optimum global. L'utilisation d'arrondis extérieurs permet un encadrement rigoureux de l'optimum global au moyen d'un ordinateur. Le représentant le plus connu de cette seconde tendance est GlobSol, le solveur basé sur l'analyse par intervalles de Kearfott [54]. Le défi était donc de combiner les avantages des deux méthodes dans une nouvelle approche à la fois efficace et rigoureuse.

Pour illustrer notre propos, considérons le problème d'optimisation suivant :

$$\begin{array}{ll} \text{minimize} & x \\ \text{subject to} & y - x^2 \geq 0 \\ & y - x^2(x-2) + 10^{-5} \leq 0 \\ & x, y \in [-10, +10] \end{array} \tag{1}$$

Ainsi que le montre la figure 1, la solution du problème 1 se trouve dans le voisinage de $x \approx 3$ et $y \approx 9$. Ce point est l'unique intersection de la courbe $y = x^2$ et $y = x^2(x-2) - 10^5$. Cependant, au point $x = 0$ et $y = 0$, les deux courbes ne sont séparées que par un ϵ arbitrairement fixé à 10^{-5}. Baron, dans ses versions 6.0 et 7.2, trouve rapidement 0 comme minimum global même lorsque la précision est fixée à 10^{-12}. Un tel défaut est particulièrement ennuyeux : ainsi que le souligne Neumaier dans [82], dans de nombreuses situations, comme des problèmes liés à la vérification de conditions de sécurité ou en chimie, la connaissance du véritable optimum

global est critique. Fournir un optimum global meilleur que l'optimum global réel peut être particulièrement ennuyeux.

Les techniques rigoureuses de relaxation linéaire que nous avons développées pour la résolution de contraintes sur le continu [65, 75, 58, 64, 60] sont à la base de l'approche que nous avons proposée pour la résolution de problèmes d'optimisation. Notre algorithme de séparation et évaluation combine l'analyse par intervalles et la programmation par contraintes au sein du schéma de résolution proposé par Horst et Huy [51]. Les techniques d'analyse par intervalles permettent l'introduction de garde-fous qui assurent la rigueur et la correction des calculs alors que la programmation par contraintes améliore les réductions de l'espace faisable.

Schématiquement, notre approche [58, 61] utilise les techniques de filtrage classiques sur l'ensemble des contraintes du problème à optimiser auquel est ajoutée une contrainte sur les bornes de la fonction objectif. Les relaxations servent de base à la détermination de la borne inférieure de la fonction objectif, les méthodes rigoureuses mises en place garantissant que la borne inférieure calculée est effectivement une borne inférieure du problème. La principale difficulté reste la recherche d'une borne supérieure, recherche qui nécessite la certification de l'existence d'une solution. Les méthodes proposées par Hansen [48] pour traiter les systèmes sous-contraints sont utilisées à cet effet, en conjonction avec des méthodes de recherche locale pour trouver un candidat à la faisabilité. Lorsqu'une région faisable est déterminée, elle est conservée parmi l'ensemble des régions faisables trouvées, et sert à déterminer la borne supérieure de la fonction objectif. Ce processus se termine lorsque la distance entre les deux bornes rigoureuses de la fonction objectif est inférieure à un epsilon donné.

Les comparaisons expérimentales [58, 61] que nous avons faites avec Baron et GlobSol ont montré que, la plupart du temps, notre approche tenait la comparaison avec Baron et se comportait bien mieux que GlobSol. Une approche similaire a été développée par Kearfott dans [56], mais les expérimentations ont montré que notre approche restait meilleure.

2.3.3 Contraintes et réduction basée sur l'optimalité

Introduite par Ryoo et Sahinidis dans [96], la réduction basée sur l'optimalité (ou OBR pour "Optimality Based Reduction") cherche à exploiter les bornes connues de la fonction objectif afin de réduire la taille des domaines des variables. Cette technique utilise une propriété bien connue du point-selle (ou point-col, cf., par exemple, le chapitre 5 de [79]) afin de calculer de nouvelles bornes du domaine d'une variable en fonction des bornes connues du domaine de la fonction objectif. Toutefois, l'algorithme de base de l'OBR n'est pas rigoureux et peut donc rendre le processus global de résolution incomplet et incapable d'atteindre effectivement l'optimum global.

Illustrons le fonctionnement de l'OBR à l'aide d'un petit exemple d'op-

timisation globale :

$$\begin{array}{ll} \text{minimize} & -x_1 + x_1x_2 - x_2 \\ \text{subject to} & \\ & \text{c1}: \ -2x_1 + 2x_2 \leq 1 \\ & \text{c2}: \ 3x_1 - x_2 \leq 3 \\ & x_1, x_2 \in [0, 5] \end{array}$$

Pour linéariser cet exemple, la variable x_3 est substituée à l'unique terme non linéaire x_1x_2. Ce terme non linéaire est linéarisé grâce à une relaxation bilinéaire due à McCornick [71]. Le système linéaire ainsi obtenu est :

$$\begin{array}{ll} \text{minimize} & -x_1 + x_3 - x_2 \\ \text{subject to} & \\ & \text{c1}: \ -2x_1 + 2x_2 \leq 1 \\ & \text{c2}: \ 3x_1 - x_2 \leq 3 \\ & \text{c3}: \ \overline{x}_2x_1 + \overline{x}_1x_2 - x_3 \leq \overline{x}_1\overline{x}_2 \\ & \text{c4}: \ \underline{x}_2x_1 + \underline{x}_1x_2 - x_3 \leq \underline{x}_1\underline{x}_2 \\ & x_1, x_2 \in [0, 5], \ x_3 \in [0, 25] \end{array}$$

Pour le problème linéarisé, l'algorithme du simplexe fournit la solution duale $\lambda_1^* = 0$ et $\lambda_2^* = 0.1666$, et $L = -1.08333$ comme borne inférieure. Le solveur local Minos [81] appliqué au problème initial donne une borne supérieure $U = -1.005$ correspondant au point faisable $x^* = (0.917, 1.062)$. L'OBR applique la relation $-2x_1 + 2x_2 - 1 \geq -(U - L)/\lambda_1^*$ à la contrainte active $c1$ pour générer la relation $x_1 \leq 4.57$ et permettre ainsi de retirer l'intervalle $[4.57, 5]$ du domaine de x_1. Le point critique est ici le calcul rigoureux de la solution duale en utilisant une implémentation efficace de la méthode du simplexe, i.e., une implémentation qui s'appuie sur des calculs avec des nombres en virgule flottante. Autrement dit, à cause des erreurs d'arrondi, l'optimum global peut être perdu lorsque l'OBR est appliquée pour réduire le domaine d'une variable x_i. Dans [55, 56], Kearfott a proposé une implémentation rigoureuse de l'OBR, implémentation basée sur un encadrement rigoureux de la solution duale. Malheureusement, cette méthode souffre de plusieurs restrictions et est lente.

Notons que les techniques suggérées par Neumaier et al [83] ne permettent pas de calculer une solution rigoureuse du problème dual.

Pour obtenir un résultat valide, les solutions duales doivent donc être prouvées afin de conserver la rigueur du processus de résolution par séparation et évaluation. Comme indiqué ci-dessus, Kearfott [55] a exploré ce problème dans le cas particulier des relaxations linéaires. Cependant, son approche soulève deux problèmes majeurs :

1. La substitution aux contraintes d'égalités par deux inéquations introduit des contraintes redondantes. Dans ce cas, la méthode de Newton qu'il utilise ne permet pas de valider les solutions.
2. Les contraintes inactives [1] sont identifiées à l'aide des solutions duales approximatives fournies par la méthode du simplexe.

Kearfott traite ces difficultés en relâchant la relaxation. Par conséquent, la méthode finale de validation basée sur la méthode de Newton appliquée à un système de Kuhn-Tucker [2] ne réussit pas toujours, et quand elle réussit, les bornes trouvées sont plutôt larges en raison du relâchement de la relaxation.

Nous avons proposé une approche rigoureuse et complète de l'implémentation de l'OBR en nous appuyant sur les outils naturels de la PPC.

Une observation essentielle est que les techniques de filtrage peuvent être utilisées comme technique de réfutation, c'est-à-dire, pour prouver qu'aucun point faisable n'est contenu dans le domaine de x_i lorsque celui-ci est réduit à $[\underline{x_i}, x_i']$. De fait, si le système de contraintes constitué de l'ensemble des contraintes du problème et d'une contrainte sur le domaine de variation de la fonction à minimiser n'a aucune solution lorsque le domaine de x est fixé à $[\underline{x_i}, x_i']$, alors la réduction calculée par la méthode de l'OBR est valide. Si le filtrage ne peut prouver l'absence de solution au sein de la boîte considérée, cette boîte doit juste être ajoutée à la liste des boîtes qui doivent encore être traitées par l'algorithme de séparation et évaluation, garantissant ainsi la perte d'aucune solution.

Nos expérimentations ont montré que la quasi-totalité de l'apport de l'OBR est perdu avec l'approche prônée par Kearfott. A contrario, l'approche basée sur la PPC, non seulement préserve les performances, mais de manière surprenante, elle les améliore : il s'avère que les réductions erronées de domaines effectuées par l'approche non rigoureuse pénalisent le calcul de la borne supérieure courante de la fonction objectif.

L'approche basée sur la PPC introduite ici est environ cinq fois plus rapide que l'approche de Kearfott. De fait, le surcoût dû à notre approche reste négligeable puisque le mécanisme de preuve s'appuie sur la $2B$-consistance, une technique de filtrage peu coûteuse [3].

Cette approche montre comment les techniques de filtrage issues de la programmation par contraintes peuvent permettre une implémentation rigoureuse et efficace de la réduction basée sur l'optimalité. Grâce à la programmation par contraintes, l'algorithme de séparation et évaluation peut bénéficier de l'apport de l'OBR à l'aide d'un simple mais efficace processus de réfutation et l'intégrer de manière naturelle et rigoureuse.

Ces travaux, détaillés dans [62, 63], soulignent l'intérêt qu'il y a à consi-

1. Les contraintes inactives sont les contraintes dont la solution duale est nulle.
2. Pour une introduction détaillée au système de Kuhn-Tucker, voir [4].
3. C'est la raison pour laquelle notre filtrage global basé sur les relaxations linéaires n'est pas mis en œuvre dans ces exemples.

dérer l'ensemble des approches possibles lorsqu'une implémentation rigoureuse d'une méthode est requise. Une construction mathématique complexe ne présente pas toujours un intérêt lorsqu'une approche plus pragmatique du problème s'intègre naturellement au processus de résolution.

2.3.4 Recherche d'une borne supérieure du minimum global

Dans un processus de séparation et évaluation [51] (ou « branch and bound »), l'un des aspects les plus critiques est de déterminer une borne supérieure du minimum global. Les approches rigoureuses doivent alors fournir une borne correspondant à un point faisable du problème, i.e., une solution dont l'existence est prouvée. La preuve d'existence d'une solution reste un processus coûteux susceptible d'échouer même lorsque la boîte testée contient effectivement des solutions. Par exemple, du fait de l'utilisation de l'arrondi extérieur, le test d'existence et d'unicité basé sur l'inclusion stricte d'une itération de l'opérateur de Newton [47] peut échouer sur machine bien qu'il soit positif mathématiquement. Pourtant, l'obtention d'une bonne borne supérieure, i.e., aussi proche que possible du minimum global, permet d'élaguer substantiellement l'arbre de recherche et d'améliorer significativement la vitesse de résolution du problème.

La recherche de cette borne supérieure, et du point faisable associé, se base le plus souvent sur une méthode locale. Ces méthodes présentent l'avantage d'être rapides et de fournir une borne supérieure souvent proche du minimum global. La solution proposée par ces méthodes locales étant calculée de manière approximative, elles sont rendues rigoureuses en construisant par inflation une boîte autour du point calculé par la méthode locale et en lui appliquant un test d'existence (un test de Borsuk dans notre cas [34]). En pratique, l'obtention d'un point pour lequel cette démarche réussit reste un processus coûteux nécessitant souvent plusieurs itérations.

Nous avons donc proposé une autre approche pour calculer des prédictions assez précises de points faisables [37], la qualité de ces prédictions conditionnant le succès de la preuve d'existence. Notre démarche consiste à exploiter la solution de la relaxation linéaire construite pour calculer une borne inférieure du minimum global. Cette solution, disponible sans coût supplémentaire après chaque calcul de la borne inférieure, présente un inconvénient : par construction, elle se trouve en dehors de l'espace faisable. Elle est cependant proche du minimum global et s'en rapproche après chaque coupe appliquée à la boîte courante. Cette proximité grandissante avec les coupes successives appliquées lors de la descente dans l'arbre de recherche combinée avec la construction d'une boîte autour de la solution du programme linéaire à chaque tentative de preuve d'existence d'une solution faisable semble une approche suffisante pour construire une borne supérieure. Nos expériences ont cependant montré l'inefficacité de ce procédé : la preuve d'existence ne fonctionne que lorsque la solution de la relaxation est assez proche de l'es-

pace faisable, phénomène qui n'arrive le plus souvent que sur des domaines très petits. Afin d'exploiter la solution du problème linéaire, une étape supplémentaire est donc nécessaire : il faut calculer, à partir de la solution du programme linéaire, une pseudo-solution, i.e., un point suffisamment proche de l'espace faisable de manière à permettre à la preuve d'existence d'opérer sur une boîte qui englobe au moins une partie de l'espace faisable.

Notre approche s'appuie sur une adaptation de la méthode de Newton [85] à des systèmes sous-contraints d'équations et d'inéquations. Lorsque les conditions sont remplies, la méthode de Newton converge par itérations successives vers une solution d'un système carré d'équations [47]. Les contraintes d'un problème à optimiser formant généralement un système sous-contraint, l'algorithme utilise alors la pseudo-inverse de Moore-Penrose [92]. Les inéquations sont traitées en les transformant en équations à l'aide de variables d'écart. Par ce simple moyen, la solution du programme linéaire est rapprochée de l'espace faisable jusqu'à atteindre une précision donnée.

Les expérimentations sur une base significative de problèmes ont montré la pertinence de cette approche. Sur ces problèmes, elle se révèle, en moyenne, plus efficace que les méthodes locales lorsqu'elles sont rendues rigoureuses. Des problèmes non résolus par les méthodes locales, que ce soit parce que la preuve a échoué ou parce que ces méthodes n'ont pu fournir une solution dans le délai imparti, l'ont été par cette simple méthode.

Une description plus complète ainsi que l'ensemble des expérimentations sont disponibles dans [37].

2.3.5 Conclusion

Avec ces travaux sur l'optimisation globale, nous avons cherché à améliorer les performances des solveurs rigoureux en nous appuyant sur un rapprochement cohérent des techniques de relaxation et des techniques basées sur l'arithmétique des intervalles. L'implémentation que nous avons faite d'un optimiseur global basé sur ces techniques a montré les bénéfices de ce rapprochement tant en termes de vitesse de résolution que de rigueur des résultats. Nous avons aussi montré comment les réductions basées sur l'optimalité pouvaient être intégrées naturellement dans un processus d'évaluation et séparation en ne faisant appel qu'aux outils de la PPC. Nous avons aussi montré comment la solution de l'optimum global de la relaxation linéaire du problème calculée par un simplexe pouvait être utilisée pour construire une solution faisable. Notons que cette démarche rigoureuse de résolution de problèmes d'optimisation globale a été reprise, par exemple, dans les travaux de Neveu et al. [84].

2.4 Traitement de contraintes universellement quantifiées

Les problèmes de satisfaction de contraintes quantifiées (QCSP) ont de nombreuses applications en ingénierie et en conception (cf. [52, 53, 9, 35, 36]). Nous nous sommes intéressés à une sous-classe de ces problèmes où les quantificateurs existentiels précèdent les quantificateurs universels :

$$\exists x \in \mathbf{x}, \forall y \in \mathbf{y}, c_1(x,y) \wedge \cdots \wedge c_p(x,y),$$

où $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_m)$ sont des vecteurs de variables, $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ et $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_m)$ sont des vecteurs d'intervalles sur des domaines continus, et les contraintes c_i sont des inéquations de la forme $f_i(x,y) \leq 0$. En plus des problèmes introduits par [9], cette classe de QCSP peut résoudre des problèmes de conception de contrôleurs robustes que l'on trouve dans de nombreux travaux [33, 52, 31, 69, 104, 30]. Une observation essentielle est que dans toutes ces applications, les solutions forment un continuum de nombres réels. Plus que par une solution isolée, l'utilisateur est intéressé par un sous-ensemble continu de $\mathbf{x}$ dont chaque élément est une solution. C'est pour cette raison que notre algorithme calcule deux ensembles, $\mathcal{I}$ et $\mathcal{B}$ (appelés respectivement *approximation intérieure* et *approximation frontière*), qui vérifient la relation suivante : $\mathcal{I} \subseteq \text{Sol} \subseteq \mathcal{I} \cup \mathcal{B}$ (où Sol représente l'ensemble des solutions). Autrement dit, $\mathcal{I}$ ne contient que des solutions, alors que $\mathcal{B}$ est la frontière des solutions, i.e., $\mathcal{B}$ contient à la fois des solutions et des non-solutions. $\mathcal{I} \cup \mathcal{B}$ contient l'ensemble des solutions et est appelé *approximation extérieure*. Notons que, à cause des limites du calcul sur machine, ces ensembles sont approximés par des unions de boîtes.

Nous avons introduit un nouvel algorithme de calcul des ensembles $\mathcal{I}$ et $\mathcal{B}$. Par contraste avec les autres approches consacrées à la résolution des QCSP, nous traitons les QCSP comme des CSP classiques où les quantificateurs universels sont traités localement, au niveau de chaque contrainte, en introduisant un CSP équivalent au QCSP à résoudre, CSP formé de contraintes $c^{\mathbf{y}}(x)$ sur les variables x (pour plus de clarté, $\mathbf{y}$, le domaine des paramètres, est mentionné explicitement). De telles contraintes sont quantifiées universellement : $c^{\mathbf{y}}(x) \equiv \forall y \in \mathbf{y}, f(x,y) \leq 0$ où les y sont appelés paramètres [4] de la contrainte et $\mathbf{y}$, les domaines de variation de ces paramètres. Le traitement des paramètres est une caractéristique essentielle de notre algorithme.

Un des avantages de cette approche est que les paramètres peuvent être traités d'une manière adaptée à chaque contrainte. Les techniques standards de résolution de contraintes doivent cependant être adaptées à cette classe de contraintes.

L'algorithme proposé alterne étapes de filtrage/contraction des domaines avec étapes de branchement de manière standard afin de rejeter les parties

4. Les y sont appelés paramètres car les variables quantifiées universellement représentent l'ensemble des valeurs possibles pour un paramètre de conception ; en d'autres mots, la contrainte doit être vraie pour toute valeur de y appartenant à $\mathbf{y}$.

de l'espace qui ne contiennent pas de solution. Au sein de ce processus s'intercale une étape d'identification des boîtes intérieures, i.e., des boîtes qui ne contiennent que des solutions. Les éléments spécifiques de cet algorithme sont les suivants :

— **La contraction de l'espace de recherche** : La contraction locale cherche à réduire le domaine $\mathbf{x}$ à partir de la contrainte $\forall y \in \mathbf{y}^i, c_i(x, y)$. Dans ce but, Benhamou et al [9] et Ratschan [94] appliquent le contracteur $Contract_{c_i(x,\mathbf{y}^i)}(\mathbf{x})$ où y est traité comme une variable quantifiée existentiellement sur le domaine $\mathbf{y}^i$. Cette approche manque cependant d'efficacité. Aussi, nous proposons un contracteur plus efficace en instanciant le paramètre y à une valeur arbitraire $\tilde{y} \in \mathbf{y}^i$. Plus formellement, le domaine de x est réduit en utilisant le contracteur $Contract_{c_i(x,\tilde{y})}(\mathbf{x})$ qui ne rejette que les parties de $\mathbf{x}$ qui ne satisfont pas $c_i(x, \tilde{y})$ et donc pas $\forall y \in \mathbf{y}^i, c_i(x, y)$. Notons que le choix de $\tilde{y}$ n'est pas stratégique. Les réductions ainsi obtenues ne prennent en compte qu'une seule contrainte à la fois. Afin d'obtenir un domaine valide pour l'ensemble des contraintes, ces réductions sont intersectées entre elles.

— **L'identification des ensembles de solutions** : De manière classique [19, 93, 103, 9, 94], l'identification des ensembles de solutions est implémentée par application des contracteurs d'intervalles à la négation des contraintes. Ici aussi cette identification opère contrainte par contrainte et seules les boîtes prouvées solutions locales de toutes les contraintes sont solutions de la totalité du CSP. L'application d'un contracteur à la négation d'une contrainte c_i permet d'obtenir un sous-domaine $\mathbf{x}^{i\prime} \in \mathbf{x}$ tel que $\mathbf{x} \backslash \mathbf{x}^{i\prime}$ ne contient que des solutions de la contrainte c_i. Donc les valeurs de $\mathbf{x}$ qui sont en dehors de tous les $\mathbf{x}^{i\prime}$, i.e., qui sont dans $\mathbf{x} \backslash (\mathbf{x}^{1\prime} \cup \ldots \cup \mathbf{x}^{p\prime})$, satisfont toutes les contraintes. Afin d'éviter une explosion du nombre de boîtes, on considère une approximation plus faible des boîtes intérieures, $\mathbf{x} \backslash \square(\mathbf{x}^{1\prime} \cup \ldots \cup \mathbf{x}^{p\prime})$, qui est ajoutée à l'ensemble des boîtes intérieures alors que les $\mathbf{x}^{i\prime}$ seront raffinés dès la prochaine itération jusqu'à atteindre une taille suffisamment petite pour être considérées comme faisant partie des approximations frontières.

— **Le traitement des domaines des paramètres** : La taille des domaines des paramètres joue un rôle critique dans l'efficacité des contracteurs utilisés lors de l'identification des solutions. La négation des contraintes utilisées lors de cette identification ne permet pas d'utiliser le même procédé que lors de la contraction de l'espace de recherche. Trois méthodes différentes sont utilisées pour améliorer l'efficacité de ces contracteurs. La première consiste à exploiter les contractions des domaines des paramètres obtenues lors de l'identification des ensembles de solutions. [9] restaure les domaines des para-

mètres à leurs valeurs initiales perdant ainsi toute opportunité de propagation de ces réductions. Néanmoins, ces réductions peuvent être propagées sans perte de solution, ce qui constitue une amélioration significative dont l'implémentation ne nécessite aucun coût supplémentaire. La seconde méthode est la plus naturelle puisqu'elle repose sur la bissection des paramètres. Notre approche est cependant un peu différente de celle utilisée dans [94] : elle consiste à substituer à $\forall y \in \mathbf{y}, c(x,y)$ la conjonction de deux contraintes, $\forall y \in \mathbf{y}^1, c(x,y)$ et $\forall y \in \mathbf{y}^2, c(x,y)$, où $\mathbf{y}^1$ et $\mathbf{y}^2$ résultent de la bissection de $\mathbf{y}$. La troisième et dernière méthode utilise la monotonie des contraintes sur le domaine considéré pour substituer aux paramètres une instance de leur domaine. En effet, si pour la contrainte $\forall y \in \mathbf{y}, f(x,y) \leq 0$, où $\mathbf{y} = [\underline{y}, \overline{y}]$, f est croissante (resp. décroissante) relativement à y sur $\mathbf{y}$, alors la contrainte quantifiée est équivalente à la contrainte non quantifiée $f(x,\overline{y}) \leq 0$ (resp. $f(x,\underline{y}) \leq 0$). L'instanciation des paramètres peut vraiment améliorer l'efficacité de l'identification des ensembles de solutions. Elle nécessite néanmoins une évaluation des dérivées, évaluation qui peut être coûteuse dans certains cas.

Les expérimentations ont montré un gain significatif de la méthode proposée sur les approches précédentes, en particulier sur Rsolver, le solveur développé par S. Ratschan. Elles montrent aussi l'intérêt de l'utilisation des dérivées pour instancier les paramètres : s'il y a parfois un peu de pertes de temps dues à l'évaluation de ces dérivées, le nombre de problèmes résolus est plus important. L'algorithme est ainsi rendu plus robuste.

L'ensemble de ces travaux est décrit plus en détail dans [38, 39].

2.5 Conclusion

Dans cette section, nous avons décrit les travaux de recherche que nous avons effectués sur le continu. Une grande partie de ces travaux reposent sur l'utilisation rigoureuse de relaxations linéaires pour la résolution de systèmes de contraintes sur les réels et d'optimisation globale d'une fonction réelle soumise à un ensemble de contraintes sur les réels. Dans le premier cas, notre approche s'appuie sur un filtrage global qui capture le sous-système linéaire augmenté d'un certain nombre de relaxations linéaires de termes non linéaires. Dans le second cas, les relaxations linéaires sont utilisées afin de déterminer une borne inférieure de la fonction objectif. Dans les deux cas, le calcul des coefficients des relaxations linéaires est effectué par des procédures rigoureuses garantes de la préservation de l'ensemble des solutions du problème initial, et le système linéaire obtenu est résolu à l'aide d'un simplexe dont le minimum global est lui aussi calculé de manière rigoureuse.

Ces recherches sont complétées par d'autres contributions qui vont de l'utilisation de sous-espaces non solutions comme critères de choix et de coupe, à l'exploitation de la solution globale du programme linéaire pour

construire une solution faisable, en passant par une implémentation rigoureuse de la réduction basée sur l'optimalité. Nous avons aussi contribué à améliorer la résolution de systèmes de contraintes quantifiés universellement.

3 Contraintes sur les flottants

3.1 Introduction

L'analyse de programmes représente un domaine d'application important pour la programmation par contraintes (PPC). Cette dernière s'y est plus particulièrement illustrée dans la génération automatique de jeux de tests [40, 14] et la vérification de la non-conformité d'un programme vis-à-vis de sa spécification [21]. Cependant, lorsqu'un programme contient des expressions effectuant des calculs sur les flottants, les approches classiques de la PPC ne permettent pas un traitement correct de ces expressions :

— L'arithmétique des flottants ne dispose pas des mêmes propriétés que l'arithmétique des réels. Par exemple, l'arithmétique des flottants n'est ni distributive, ni associative. Les solveurs de contraintes sur les réels ou sur les rationnels ne conservent donc pas l'ensemble des solutions sur les flottants.

— La taille des domaines est considérable et ceux-ci ne sont pas uniformément répartis (e.g., la moitié des flottants sont contenus dans l'intervalle $[-1.0, 1.0]$)). Les techniques classiques de filtrage comme la consistance d'arc sont donc totalement inadaptées à la manipulation de tels domaines.

C'est pour ces raisons que nous avons dû développer de nouvelles techniques capables de traiter correctement les flottants. Pour ce faire, nous avons introduit un nouveau type de contraintes qui capture correctement les calculs sur les flottants et dont les domaines sont des intervalles de flottants. Naturellement, nous avons aussi introduit un ensemble de techniques propres à résoudre de tels CSP en conservant l'ensemble des solutions de ces calculs. L'ensemble des travaux présentés dans cette section traitent de ces questions.

3.2 Un cadre pour les contraintes sur les flottants

Dès le début du projet Inka [5], le problème que nous devions traiter était celui des contraintes sur les flottants. Si la problématique avait été clairement identifiée, à notre connaissance aucune solution n'était alors disponible. Nous avons donc développé le premier solveur de contraintes sur les flottants.

Pour construire un solveur de contraintes sur les flottants, le premier facteur à prendre en compte est la taille des domaines. Les domaines des problèmes traités sont très larges et, le plus souvent, aucune information

5. Le projet RNTL Inka (2000-2002) avait pour objet la génération automatique déterministe de données de test selon des critères de couverture structurelle.

n'étant disponible, ils couvrent la totalité des flottants. Une représentation par intervalles des domaines considérés semble bien convenir à cette situation. Les domaines de valeurs y sont donc représentés par des intervalles de flottants à bornes dans $\mathcal{F}$: $x \in \mathbf{x} = [\underline{x}, \overline{x}] = \{x \in \mathcal{F}, \underline{x} \leq x \leq \overline{x}\}$ avec $\underline{x} \in \mathcal{F}$ et $\overline{x} \in \mathcal{F}$.

La taille des domaines exclut, au moins dans un premier temps, l'utilisation d'une consistance d'arc [68] et oriente vers des filtrages construits autour de consistances limitées aux bornes des domaines.

De prime abord, les solveurs de contraintes sur les réels semblent fournir les outils nécessaires au traitement des contraintes sur les flottants. Ils partagent avec ces derniers des domaines basés sur des intervalles à bornes dans $\mathcal{F}$ et les filtrages sur les réels font largement appel à une arithmétique des intervalles basée sur des arrondis extérieurs afin de garantir la conservation des solutions sur $\mathcal{R}$. Pourtant un tel choix ne résiste pas longtemps à l'analyse : les nombreuses limites de l'arithmétique des flottants invalident la plupart des méthodes habituellement utilisées pour résoudre un problème de contraintes sur le continu.

La $2B$-consistance [66], qui repose sur une propriété locale restreinte aux bornes des domaines d'une variable au niveau d'une unique contrainte, nécessite le calcul de la projection d'une contrainte c_k sur la variable x_i dans l'espace délimité par l'ensemble des variables qui apparaissent dans c_k privé de la variable x_i. Le calcul de cette projection étant impossible dans le cas général, les filtrages par $2B$-consistance s'appuient sur une décomposition de la contrainte initiale en contraintes élémentaires binaires ou ternaires pour lesquelles le calcul des fonctions de projections est trivial [18]. Malheureusement, les fonctions de projections inverses, i.e., celles qui consistent à projeter la contrainte sur l'une de ses variables d'entrées, ne sont pas conservatives des solutions sur les flottants. Par exemple, la projection inverse de la contrainte $16.0 = 16.0 + x$ sur la variable d'entrée x est obtenue par les solveurs sur les réels en calculant l'expression $[16.0 - 16.0, 16.0 - 16.0]$, qui, même avec l'emploi d'arrondis extérieurs, génère un intervalle singleton $[0.0, 0.0]$. Or, sur les flottants [6], tous les flottants doubles contenus dans l'intervalle $[-8.88178419700125232e-16, 1.77635683940025046e-15]$ sont solutions de la contrainte.

La Box-consistance [10] est une relaxation de la consistance d'arc moins fine que ne l'est la $2B$-consistance. Elle génère pour chaque variable d'une contrainte des relations univariées en remplaçant toutes les autres variables par l'intervalle qui représente leur domaine respectif. Mais, contrairement à la $2B$-consistance, elle ne nécessite aucune décomposition de la contrainte initiale. Les implémentations de la Box-consistance utilisent un Newton par intervalles afin de calculer le zéro le plus à gauche et le zéro le plus à droite de la relation univariée qu'elle a générée. Là aussi, des solutions

6. Pour des doubles et avec un arrondi au plus près pair.

sur les flottants peuvent être perdues à cause des formes de Taylor introduites par le Newton par intervalles. Par exemple, considérons l'équation $f(x, y, z) = x + y + z = 0$, avec $x \in \mathbf{x} = [-1, 1]$, $y \in \mathbf{y} = [16.0, 16.0]$ et $z \in \mathbf{z} = [-16.0, -16.0]$. Une simple itération de l'opérateur de Newton $\mathbf{x} \leftarrow \mathbf{x} \cap (m(\mathbf{x}) - f(m(\mathbf{x}),\mathbf{y},\mathbf{z})/\frac{\partial f}{\partial x}(\mathbf{x},\mathbf{y},\mathbf{z}))$ a pour résultat $\mathbf{x} = [0, 0]$, un sous-ensemble très restrictif de l'ensemble des solutions sur les flottants.

A priori, aucune des consistances sur intervalles disponibles ne semble convenir au traitement correct des contraintes sur les flottants. Pourtant, un examen plus attentif de la *Box*-consistance montre que l'utilisation d'un Newton par intervalle n'est qu'un moyen, certes efficace, de calculer les zéros de la fonction considérée. Et, la définition de cette consistance ne mentionne pas d'algorithme particulier.

Pour construire un substitut au Newton par intervalles, une observation clef est que l'arithmétique des intervalles est conservative de l'ensemble des solutions sur les flottants [7] pour autant que l'ordre d'évaluation soit identique à celui des opérations correspondantes du programme analysé. Puisque cette arithmétique est conservative, alors pour tout sous-domaine $\mathbf{x}' = [\underline{x}, x']$ avec $\underline{x} < x' < \overline{x}$, si $0 \notin f(\mathbf{x}')$ alors $\mathbf{x}'$ ne contient pas de solution pour f et $\mathbf{x}$ peut être réduit à $[x'^+, \overline{x}]$. Ce principe de réfutation est suffisant pour construire un algorithme de « shaving » capable de maintenir la consistance de contraintes sur les flottants. Un algorithme classique de propagation construit sur ces réductions permet de réduire l'ensemble des domaines du problème en ne propageant que les réductions supérieures à un ϵ donné.

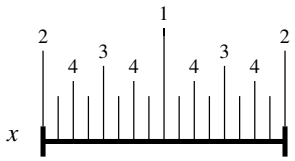

FIGURE 2 – Schéma d'énumération barycentrique

L'ensemble des flottants étant grand, mais énumérable, une procédure d'énumération barycentrique (cf. figure 2) permet l'obtention de solutions du problème. Elle est paramétrée par une profondeur d'énumération. Cette procédure est appliquée par permutation circulaire à l'ensemble des variables. Naturellement, chaque instanciation de variable est suivie d'un filtrage. En théorie, si la profondeur d'exploration est suffisante, la recherche de solutions est complète.

Dans [77], nous détaillons ces travaux, fixons un tout premier cadre pour les contraintes sur les flottants et proposons une première approche pour le filtrage et la résolution de ces contraintes. C'est, à notre connaissance, le tout

7. Pour une arithmétique calculée avec des doubles et un arrondi extérieur et pour des contraintes sur des flottants doubles.

premier solveur de contraintes sur les flottants. Bien que le procédé de réduction des domaines mis en place soit relativement lent, nos expérimentations ont montré sa capacité à résoudre de petits problèmes sur les flottants.

3.3 Vers une consistance d'intervalles

Les travaux précédents (section 3.2) introduisent la notion de contraintes sur les flottants, fixent un cadre pour ces contraintes et proposent une première approche pour résoudre ce type de contraintes qui démontre la faisabilité d'un solveur de contraintes sur les flottants. Cependant, en s'appuyant sur un algorithme de « shaving » pour réduire les domaines des variables, ce type de filtrage manque quelque peu d'efficacité. Une démarche plus pertinente est possible si, en s'orientant vers un filtrage de type $2B$-consistance, un moyen de calculer les fonctions de projections est mis en place [73].

Une première observation utilisée dans les travaux précédents est que le calcul d'intervalles est conservatif des solutions sur les flottants. Lorsque les bornes des intervalles sont calculées avec des arrondis extérieurs, il est conservatif quel que soit le mode d'arrondi. Mais, grâce à la monotonie du mode d'arrondi [8], il reste conservatif lorsque les bornes sont calculées avec le même mode d'arrondi que celui appliqué pour l'opération considérée. Cette simple propriété permet de calculer l'ensemble des réductions de domaines qui suivent le sens naturel du calcul. Par exemple, pour une expression telle que $z = x + y$, le calcul d'une réduction du domaine de z à partir de ceux de x et y en tenant compte du mode d'arrondi courant r devient possible : $\mathbf{z} \leftarrow \mathbf{z} \cap (\mathbf{x} +_r \mathbf{y}) = \mathbf{z} \cap [\underline{x} +_r \underline{y}, \overline{x} +_r \overline{y}]$.

La principale difficulté survient lorsqu'il s'agit non plus de calculer les projections qui suivent le sens naturel du calcul, mais celles qui suivent le sens inverse. On parle alors de projections inverses par opposition aux projections précédentes qualifiées de directes. Notons qu'ainsi le sens du calcul est nécessairement capturé au niveau le plus élémentaire du filtrage et que, par conséquent, le processus de décomposition des contraintes initiales en contraintes élémentaires doit prendre en compte cet élément.

Les projections inverses s'appuient sur l'arrondi correct, une propriété garantie par la norme IEEE, au moins pour les opérations principales : une fonction sur les flottants $f_{\mathcal{F}}$ est correctement arrondie si et seulement si $f_{\mathcal{F},r}(x) = \circ_r(f(x))$. En d'autres mots, une fonction sur les flottants est correctement arrondie si elle est équivalente à un calcul du résultat exact sur les réels avant d'arrondir ce résultat au flottant le plus proche selon la direction d'arrondi choisi. À partir de cette unique propriété, le calcul de la projection inverse est rendu possible. Par exemple, pour un arrondi vers $+\infty$, et une fonction f croissante, cette projection donne $\mathbf{x} \leftarrow \mathbf{x} \cap [\circ^+(f^{-1}(\underline{y}^-)), \circ_{-\infty}(f(\overline{y}))]$ où $\circ^+()$ est un arrondi à $+\infty$ si le résultat est inexact (i.e., $f^{-1}(y)$ n'est pas un flot-

8. i.e., si $x \in \mathcal{R}$ et $y \in \mathcal{R}$ avec $x \leq y$ alors $\circ(x) \leq \circ(y)$.

```
int solve_cubic (double a, double b, double c)
{
  double q = (a * a - 3 * b);
  double r = (2 * a * a * a - 9 * a * b + 27 * c);

  double Q = q / 9;
  double R = r / 54;

  double Q3 = Q * Q * Q;
  double R2 = R * R;

  double CR2 = 729 * r * r;
  double CQ3 = 2916 * q * q * q;

  if (R == 0 && Q == 0)
    ...  /* Point a atteindre */
  else
    ...
  ...
```

FIGURE 3 – `solve_cubic` extrait de la Gnu Scientific Library

tant) et, le successeur de $f^{-1}(y)$ si le résultat est exact (i.e., $f^{-1}(y) \in \mathcal{F}$). Ce résultat ne dépendant que de l'arrondi correct, du mode d'arrondi et de la monotonie de la fonction, il couvre naturellement tous les formats de flottants qui respectent ces conditions.

Couplé à une recherche de solutions basée sur une bissection des domaines des variables, ce filtrage permet la construction d'un solveur de contraintes sur les flottants complet à même de trouver toutes les solutions d'un problème de contraintes sur les flottants. Nos expérimentations ont montré la viabilité de cette approche qui se montre notablement plus efficace que la version précédente basée sur la *Box*.

L'ensemble, projections directes et projections inverses, sert de base à la construction d'une adaptation de la $2B$-consistance aux flottants. Le filtrage obtenu se montre suffisamment fort pour, au moins dans certains cas, propager immédiatement l'instanciation d'une variable d'entrée sur les autres variables d'entrées, comme dans l'exemple de `solve_cubic` (cf. figure 3) où un simple filtrage sur une instanciation de $a = 1.5$ permet de déduire que $b = 0.75$ et $c = 0.125$ pour attendre la première branche du `if`.

J'ai introduit cette technique de filtrage dans [73]. En collaboration avec Bernard Botella et Arnaud Gotlieb, nous l'avons ensuite étendue dans le contexte de l'évaluation symbolique [13], en y détaillant, entre autres choses, les cas particuliers que sont les infinis et les zéros signés. Ce filtrage sur les flottants est à la base de FPCS, le solveur de contraintes sur les flottants que j'ai développé.

3.4 Amélioration grâce à une propriété de la soustraction

Bien que les fonctions de projection introduites dans [73] améliorent notablement le filtrage des contraintes sur les flottants, elles ne sont pas exemptes de certains défauts. En testant ce type de filtrage sur des exemples variés, des cas de convergence lente se sont manifestés. Un examen détaillé du fonctionnement du filtrage a mis en évidence un problème de convergence lente particulièrement néfaste dans le cas de l'addition et de la soustraction, cas d'autant plus curieux qu'il se produit au sein d'une même opération.

Pour illustrer notre propos, considérons la contrainte $z = x + y$ avec un arrondi à $+\infty$ et où $\mathbf{z}$ est réduit à l'intervalle dégénéré $[2^-, 2^-]$, i.e., le prédécesseur de 2. Si $\mathbf{x}$ et $\mathbf{y}$ valent tous deux $[-100.0, 100.0]$, alors, la première projection directe de x et y sur z donnera $\mathbf{z} \leftarrow \mathbf{z} \cap [-200.0, 200.0] = [2^-, 2^-]$. Le domaine de z étant un intervalle dégénéré, aucune réduction n'en est possible. Avec les projections inverses, on obtient : $\mathbf{x} \leftarrow \mathbf{x} \cap [-98.0, 102.0^-] = [-98.0, 100.0]$ et $\mathbf{y} \leftarrow \mathbf{y} \cap [-98.0, 100.0^-] = [-98.0, 100.0^-]$. Deux observations peuvent être faites à ce stade. La première est que les domaines calculés pour x et y sont surapproximés, la valeur -98.0 n'ayant pas de support dans z. La seconde observation est que la borne supérieure de $\mathbf{y}$ est inférieure d'un flottant à celle de $\mathbf{x}$. C'est dans ce dernier phénomène que la convergence lente observée prend ses racines. En effet, à la seconde itération du calcul des projections inverses, le même phénomène va se reproduire, diminuant chacune des bornes de $\mathbf{x}$ et $\mathbf{y}$ d'un flottant, avec $\mathbf{x} \leftarrow \mathbf{x} \cap [-98.0^-, 100.0^-] = [-98.0^-, 100.0^-]$ et $\mathbf{y} \leftarrow \mathbf{y} \cap [-98.0^-, (100.0^-)^-] = [-98.0^-, (100.0^-)^-]$. De fait, le filtrage réduit les bornes de $\mathbf{x}$ et $\mathbf{y}$ d'un flottant à chaque itération, processus qui prend plusieurs heures, avant d'atteindre pour point fixe, $\mathbf{x} = [-2.0^-, 4.0^-]$ et $\mathbf{y} = [-2.0^-, 4.0^-]$, dont les bornes sont des supports.

Un moyen bien connu d'éviter de telles convergences lentes est de stopper le calcul du point fixe avant terme en prenant comme condition d'arrêt de la boucle la taille des réductions obtenues sur les domaines. Si cette approche est volontiers adoptée sur les réels, où les convergences lentes ont d'autres origines, elle ne convient pas ici : les domaines de x et y sont trop largement surapproximés pour permettre un fonctionnement efficace du solveur en utilisant un arrêt prématuré du filtrage.

Pour remédier à ce problème et accélérer le fonctionnement du filtrage, nous avons introduit une nouvelle propriété sur les flottants. Elle se base sur l'observation suivante : étant donné un flottant positif non nul x, la plus petite valeur non nulle que l'on puisse produire par soustraction est $x - x^-$. Cette propriété de la soustraction déduite de cette observation est la suivante : soit $z = x \ominus y$, où z est un flottant positif non nul et inférieur à l'infini tel que z s'écrive en binaire $1.b_2 \cdots b_i 0 \cdots 0 * 2^{e_z}$ avec $b_i = 1$, soient $\alpha = 1.1 \cdots 1 * 2^{e_z + nb_z}$ avec $nb_z = p - i$, où p est la taille en bits du signifiant, et $\beta = \alpha \oplus z$, alors il n'y a pas de $x \in \mathcal{F}_p$, $x > \beta$, ni de $y \in \mathcal{F}_p$, $y > \alpha$ tel que $x \ominus y = z$. De plus, $\beta \ominus \alpha = \beta - \alpha = z$, i.e., ce calcul est exact et donc

indépendant du mode d'arrondi.

Cette propriété qui donne un maximum pour x et y dans le cas où $0 < z < +\infty$, s'étend par symétrie à $-\infty < z < 0$ et à l'addition. Elle s'étend aussi aux intervalles en recherchant dans l'intervalle $\mathbf{z}$ le flottant ayant le plus de zéros consécutifs de poid faible. En combinant ce résultat avec le calcul des projections inverses, le filtrage obtenu devient beaucoup plus efficace. Par exemple, en l'appliquant au cas pris comme exemple illustratif, le filtrage produit immédiatement le résultat optimal sans nécessiter de longues heures de calcul.

Ces travaux, menés en collaboration avec Bruno Marre du CEA, sont décrits plus en détail dans [70]. Ils ont permis d'améliorer notablement le fonctionnement des solveurs sur les flottants, en particulier, de celui développé au CEA par Bruno Marre et de FPCS, le solveur que j'ai développé.

3.5 Et si l'on plongeait le problème dans $\mathcal{R}$?

Raisonner sur les flottants est une tâche d'autant plus ardue que les propriétés de l'arithmétique des flottants sont pauvres. Elles interdisent les manipulations symboliques habituelles qui tendent à simplifier les expressions à traiter. Ainsi, alors que nombre d'occurrences multiples disparaissent sous l'effet de ces manipulations, les contraintes sur les flottants doivent en conserver la plupart afin de respecter l'ordre des opérations, ordre dont dépend le résultat de l'évaluation de ces expressions.

Les filtrages forts tels que les kB-consistances [66] offrent un moyen bien connu de traiter le problème des occurrences multiples. Et, sur la base des consistances sur les flottants introduites précédemment, il est tout à fait possible de calculer ces consistances fortes. Il y a cependant un certain nombre de limites. Les consistances fortes comme la kB reposent sur une $(k-1)B$. Or les filtrages de base peinent déjà à passer à l'échelle. Un filtrage fort qui serait basé sur les filtrages de base ne ferait qu'empirer ce problème.

Au regard de l'ensemble des limitations liées aux flottants et à l'ensemble des méthodes de résolutions disponibles sur les réels, il semble naturel de vouloir plonger les problèmes de contraintes sur les flottants dans les réels. Les bénéfices attendus d'une telle démarche semblent nombreux et prometteurs. C'est pourquoi nous avons proposé à Said Mohammed Belaid, dans le cadre de sa thèse, de travailler sur une approximation sur les réels des contraintes sur les flottants.

Avant d'aller plus dans les détails, regardons ce qui se passe sur un petit exemple : soit la contrainte $z = x \oplus y \ominus x$. Sur les réels, cette expression se simplifie naturellement en $z = y$. Ce n'est pas le cas sur les flottants. Par exemple, sur les flottants et avec un mode d'arrondi au plus près, le calcul de $10.0 \oplus 10.0^{-8} \ominus 10.0$ ne donne pas 10.0^{-8} comme sur les réels mais 0 du fait de l'absorption de 10.0^{-8} par 10.0 lors de l'addition. Les simplifications valides sur les réels ne s'appliquent donc pas aux flottants. Maintenant, supposons

que $x \in [0.0, 10.0]$, $y \in [0.0, 10.0]$ et $z \in [0.0, 10.0^8]$. Un filtrage sur les flottants basé sur une consistance de type $2B$ va certes réduire le domaine de z à $[0.0, 20.0]$, mais sans pouvoir aller plus loin. Il n'est pour autant pas capable de réduire les domaines de x et y. Les limites de ce filtrage sont liées à l'incapacité de la $2B$ de traiter correctement les occurrences multiples. Un filtrage plus fort de type $3B$ a lui la capacité de réduire le domaine de z à $[0.0, 10.01835250854492188]$. Mais il ne réussira pas à réduire le domaine de z si x et y apparaissent plus de deux fois dans l'expression à calculer comme dans la contrainte $z = x \oplus y \ominus x \ominus y \oplus x \oplus y \ominus x$.

L'approche introduite dans [7] s'appuie sur une relaxation non linéaire dans $\mathcal{R}$ des contraintes élémentaires sur les flottants. La contrainte $z = x \oplus y \ominus x$ est tout d'abord décomposée en un système de contraintes élémentaires équivalentes : $tmp_1 = x \oplus y \wedge tmp_2 = tmp_1 \ominus x \wedge z = tmp_2$. Chaque contrainte élémentaire est alors approximée par des inéquations sur les réels, inéquations qui peuvent contenir des termes non linéaires :

$$\begin{cases} (1 - \frac{2^{-p}}{(1-2^{-p})})(x+y) \leq tmp_1 \\ tmp_1 \leq (1 + \frac{2^{-p}}{(1+2^{-p})})(x+y) \\ (1 - \frac{2^{-p}}{(1-2^{-p})})(tmp_1 - x) \leq tmp_2 \\ tmp_2 \leq (1 + \frac{2^{-p}}{(1+2^{-p})})(tmp_1 - x) \\ z = tmp_2 \end{cases}$$

où p est la taille en bits du signifiant des flottants utilisés. Lorsque cela est nécessaire, les termes non linéaires sont ensuite approximés par des relaxations linéaires à la manière de celle utilisées dans [64]. Le système résultant est donc un système d'équations et d'inéquations linéaires qu'un solveur linéaire [9] comme le simplexe sait résoudre. En maximisant et minimisant chacune des variables du problème initial sur les flottants, il est possible d'en réduire les domaines. Par exemple, une maximisation sur la variable z permet d'en réduire le domaine à $z \in [0, 10.0000023841859]$. Notons que, contrairement à la $3B$, cette approche permet de réduire le domaine de z y compris lorsque le nombre d'occurrences est supérieur à 2 comme dans la contrainte $z = x \oplus y \ominus x \ominus y \oplus x \oplus y \ominus x$.

La démarche proposée s'appuie donc sur un ensemble de relaxations dans les réels des opérations élémentaires sur les flottants. Chaque relaxation dépend du mode d'arrondi utilisé, de la taille du signifiant, et de la nature du résultat de l'opération à savoir, négatif ou positif et, normalisé ou dénormalisé. Par exemple, pour un arrondi à $-\infty$ et un résultat positif normalisé et inférieur à $max_{\mathcal{F}}$, la plus grande valeur numérique positive de $\mathcal{F}$, l'approximation utilisée est $\frac{1}{1+2^{-p+1}}(x \cdot y) < x \odot y \leq (x \cdot y)$ où $\odot$ et $\cdot$ représentent la même opération binaire, respectivement, sur les flottants et sur les réels.

9. Les mêmes procédures que dans [75, 64] sont utilisées afin de s'assurer de la validité du résultat.

Des relaxations rassemblant les différents cas sont toujours possibles et utilisées lorsque seul le mode d'arrondi est connu : pour un mode d'arrondi à $-\infty$, et des flottants à valeurs numériques, i.e., $-max_{\mathcal{F}} < x \odot y < max_{\mathcal{F}}$, on a $z_r - 2^{-p+1}|z_r| - min_{\mathcal{F}} \leq x \odot y \leq z_r$ où $z_r = x \cdot y$ et, $min_{\mathcal{F}}$, est la plus petite valeur numérique positive non nulle de $\mathcal{F}$. La valeur absolue qui apparaît dans ces formules doit être linéarisée lorsque le problème est résolu par un programme linéaire. Ces dernières relaxations sont plus lâches que les précédentes et ne sont utilisées qu'en dernier ressort.

L'algorithme de résolution exploré est identique à celui utilisé pour les réels dans [64]. Les résultats les plus intéressants ont été obtenus dans le cas d'exemples où plusieurs variables présentaient un certain nombre d'occurrences multiples. En l'absence d'occurrences multiples, une simple $2B$ réduit autant les domaines des variables.

Plonger les contraintes sur les flottants dans les réels ouvre une multitude de possibilités susceptibles d'améliorer la résolution de contraintes sur les flottants. L'approche à base de programmation linéaire adoptée ici est clairement inspirée de nos travaux précédents sur les réels [64]. Bien que les performances obtenues soient mitigées, la même démarche devrait fournir de meilleurs résultats en optimisation. Et, bien sûr, tout algorithme sur les réels capable de résoudre efficacement un système d'inéquations pourrait bénéficier aux contraintes sur les flottants. Notons aussi qu'elle permet de plonger au sein d'un même algorithme de résolution l'ensemble des types présents dans un programme.

Ces travaux sont présentés plus en détail dans la thèse de Mohammed Said Belaid [5] que j'ai co-encadré.

3.6 Coopération entre interprétation abstraite et PPC

En parallèle au développement des contraintes sur les flottants, d'autres approches ont vu le jour. L'une des plus connues pour l'analyse statique de programmes sur les flottants reste l'interprétation abstraite [23]. Citons par exemple les travaux d'Eric Goubault et Sophie Putot sur la précision des calculs sur les flottants [42, 43] avec, en particulier, l'introduction à cette fin des zonotopes, ou l'adaptation de domaines faiblement relationnels aux flottants par Antoine Minet [80, 78] et ses travaux sur les domaines polyhédraux [16].

L'idée à la base de l'interprétation abstraite est qu'il n'est pas nécessaire de calculer les domaines concrets pour vérifier certaines propriétés d'un programme. Un exemple emblématique illustrant ce propos est l'analyse du signe d'une expression arithmétique qui fait abstraction des valeurs concrètes calculées par le programme en manipulant des domaines restreints à $+, -, 0$. Il résulte de ces abstractions un bon comportement de cette analyse qui a montré sa capacité à passer à l'échelle [24]. Cependant, l'interprétation abstraite a tendance à fournir des approximations trop lâches qui se traduisent

par la production de fausses alarmes, i.e., des cas pour lesquels on ne peut être sûr qu'un problème se produise effectivement.

Utilisée dans le cadre de l'analyse de programmes, la programmation par contraintes, si elle passe moins bien à l'échelle, fournit des approximations plus précises. Elle a aussi la capacité de fournir des contre-exemples sous la forme de jeux de valeurs pour les variables d'entrée du programme pour lesquels le problème identifié se produit. La différence la plus notable entre ces deux techniques est dans le traitement des boucles : la programmation par contraintes utilise des techniques de dépliage de boucles qui ne lui permettent pas de calculer une approximation de l'ensemble des exécutions possibles d'une boucle, alors que l'interprétation abstraite a développé un ensemble de techniques qui permettent, souvent, d'obtenir une bonne approximation de l'ensemble de ces exécutions.

Interprétation abstraite et programmation par contraintes partagent aussi quelques défauts communs comme, par exemple, une certaine sensibilité à la forme syntaxique des contraintes. Notons que cette sensibilité est d'autant plus critique sur les flottants où la plupart des manipulations symboliques qui pourraient y remédier ne peuvent être utilisées sans en modifier le résultat. Cependant, là où l'interprétation abstraite va montrer quelques limites à bien approximer une conditionnelle (cas des Zonotopes dans Fluctuat), la programmation par contraintes permet une meilleure approximation et là où la programmation par contraintes va souffrir de la présence d'occurrences multiples, l'interprétation abstraite fournit une approximation plus fine.

Par leurs approches différentes, leurs avantages et leurs défauts respectifs, ces deux techniques sont clairement complémentaires. À plusieurs reprises, l'interprétation abstraite a fait son intrusion dans le monde des contraintes, comme dans le traitement des boucles à base de techniques d'interprétation abstraite [27], ou l'intégration de domaines abstraits pour résoudre des problèmes sur le continu [102, 86].

Pour aller plus loin dans l'exploration des interactions possibles entre interprétation abstraite et programmation par contraintes, nous avons proposé dans [88, 89] un autre schéma de coopération entre ces deux techniques où interagissent des solveurs de contraintes sur les flottants et les réels avec un analyseur par interprétation abstraite afin d'analyser un programme sur les flottants. Les différents outils coopèrent par leurs domaines et sont utilisés pour explorer les différents chemins possibles. Le principal écueil est le nombre de chemins à explorer. Pour limiter l'explosion combinatoire, l'analyse se fait d'un point de jonction au point de jonction suivant en procédant à une union des domaines dépendant de chacun des chemins qui mènent d'un point de jonction donné à son suivant. L'analyse continue à partir de ces unions de domaines jusqu'au prochain point de jonction en explorant tous les chemins qui mènent du point de jonction courant au prochain point de jonction suivant jusqu'à atteindre la fin du programme. Les boucles sont dépliées jusqu'à 10 fois afin de permettre à la PPC d'affiner les domaines avant

de laisser l'IA approximer l'ensemble des exécutions possibles de la boucle restante.

Cette exploration reste correcte (elle ne perd pas de solution) mais donne une approximation moins bonne que celle qu'aurait permise une exploration de l'ensemble des chemins d'exécution. Elle reste un bon compromis en termes de précision tout en permettant de traiter des programmes plus conséquents.

RAICP, l'implémentation de cette coopération, a permis de montrer, y compris sur un exemple industriel, l'intérêt de cette approche. Sur les exemples testés, les défauts respectifs de l'interprétation abstraite et de la programmation par contraintes disparaissent le plus souvent du fait de cette coopération. Et si RAICP est naturellement plus lent que FLUCTUAT, il élimine les fausses alarmes que ce dernier avait laissées.

Ces travaux ont permis de souligner la complémentarité de ces deux approches où l'interprétation abstraite apporte ses différents domaines abstraits, sa gestion des boucles et de certaines occurrences multiples de variables et la programmation par contraintes, de meilleures approximations, sa gestion des non-linéarités et ses capacités de réfutation aussi utilisable pour vérifier la conformité des programmes vis-à-vis de propriétés.

Ces recherches se prolongent au sein de l'ANR Coverif qui rassemble différents acteurs de l'interprétation abstraite et de la programmation par contraintes afin d'explorer une collaboration plus fine de ces deux techniques au sein d'un même environnement.

3.7 Génération de cas de test d'intervalles suspects

Les programmes sur les flottants dérivent souvent de modèles mathématiques ou sont écrits avec à l'esprit la sémantique des réels. Cependant, pour des valeurs d'entrées données, le chemin parcouru par un programme sur les flottants peut sensiblement différer de celui parcouru par le même programme qui adopterait la sémantique des réels. En pratique, cela se révèle vraiment critique si, pour certaines valeurs des variables d'entrées, le résultat d'une séquence d'opérations sur les flottants diffère significativement du calcul effectué avec une séquence d'opérations équivalentes mais selon la sémantique idéalisée des réels. Par conséquent, le chemin parcouru sur les flottants peut sensiblement différer de celui parcouru sur les réels avec le même jeu de valeurs des variables d'entrées. Identifier de tels jeux de valeurs est très important pour les systèmes critiques.

Les outils d'analyse de l'erreur par interprétation abstraite [16, 25, 44] calculent une surapproximation des erreurs dues aux calculs sur les flottants. Si des comportements inappropriés peuvent être identifiés, ces surapproximations ne permettent pas de savoir si le comportement redouté se produira effectivement. La figure 4 décrit ce problème : x est une variable de décision, $[\underline{x}_{\mathcal{R}}, \overline{x}_{\mathcal{R}}]$ dénote son domaine sur $\mathcal{R}$ et $[\underline{x}_{\mathcal{F}}, \overline{x}_{\mathcal{F}}]$ dénote la surapproximation

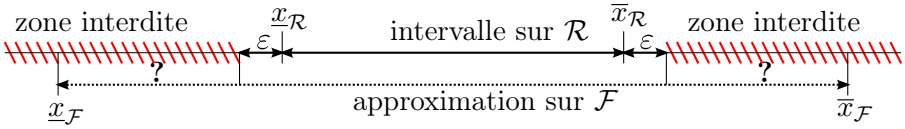

FIGURE 4 – Le programme peut-il effectivement atteindre une zone interdite sur les flottants ?

de x sur $\mathcal{F}$. En pratique, les limites des valeurs correspondant à une séquence d'opérations sur les réels peuvent être déterminées par calcul ou par les limites physiques. Une tolérance ϵ autour de ces limites est habituellement acceptée pour prendre en compte les erreurs d'approximations telles que celles dues aux mesures, aux statistiques ou même aux calculs sur les flottants. En d'autres termes, cette tolérance, spécifiée par l'utilisateur, définit une perte acceptable de précision entre les calculs effectués sur les flottants et les calculs effectués sur les réels. Toutefois, les valeurs situées en dehors de l'intervalle $[\underline{x}_{\mathcal{R}} - \epsilon, \overline{x}_{\mathcal{R}} + \epsilon]$ peuvent se traduire par un comportement erroné du programme, e.g. par prendre la mauvaise branche du flot de contrôle et donner un signal de commande totalement inapproprié. Les valeurs d'une approximation sur $\mathcal{F}$ situées en dehors de cet intervalle définissent ce que nous nommons un intervalle *suspect*.

Le problème est donc de savoir si un programme peut effectivement produire des valeurs appartenant aux intervalles suspects $[\underline{x}_{\mathcal{F}}, \underline{x}_{\mathcal{R}} - \epsilon[$ et $]\overline{x}_{\mathcal{R}} + \epsilon, \overline{x}_{\mathcal{F}}]$. Pour résoudre ce problème, nous avons introduit une nouvelle approche basée sur les contraintes à même de générer des cas de tests atteignant les intervalles suspects pour un programme sur les flottants. Notre approche consiste à réduire ce problème de génération de cas tests à un problème de résolution de contraintes sur les flottants où le domaine d'une variable critique de décision a été restreint à un intervalle suspect. Lorsqu'aucun cas de test n'a pu être généré, l'intervalle suspect est écarté.

Les résultats préliminaires des expérimentations que nous avons menées sont encourageants : ils montrent que notre approche permet la génération de cas de test pour des valeurs suspectes se trouvant en dehors de l'ensemble des valeurs acceptables sur de petits programmes, et donc de déterminer si un test critique est stable ou non. Ils montrent aussi que FPCS est plus rapide que CDFL [32] et CBMC [17].

Plus de détails sur ces travaux peuvent être trouvés dans [20].

3.8 Conclusion

Dans cette section, nous avons décrit les recherches que nous avons menées sur les contraintes sur les flottants. Ce type de contraintes particulières n'avait pas d'existence avant ces travaux. Elles sont pourtant nécessairement pour traiter des expressions faisant appel à des calculs sur les flottants, chose à laquelle de plus en plus de programmes font appel.

Ces recherches sur les flottants partent de l'introduction d'un cadre pour les traiter correctement [77], pour s'attacher ensuite à améliorer le fonctionnement d'un solveur sur les flottants, que ce soit en introduisant une forme de filtrage plus performante [73, 13] basée sur une consistance de type $2B$, en améliorant le fonctionnement de projections particulières telle que l'addition [70], en plongeant les contraintes sur les flottants dans les réels [64], ou en faisant collaborer contraintes sur les flottants et interprétation abstraite [88, 89]. Une application à la détection de fausses alarmes illustre le fonctionnement des outils développés sur un exemple industriel [89].

4 Conclusion et perspectives

Les travaux décrits dans ce document couvrent pour l'essentiel mes recherches sur les techniques de résolution de problèmes sur le continu et mes recherches sur les contraintes sur les flottants, domaine dont j'ai posé les fondements.

Sur le continu, l'introduction des techniques de relaxation linéaire à des fins de résolution de problèmes de contraintes ou d'optimisation globale sous contraintes a, en fournissant un schéma correct de mise en œuvre, montré l'intérêt effectif de ces approches. Divers travaux, principalement concentrés sur leur utilisation dans le contexte de l'optimisation globale, ont suivi les nôtres, que ce soient, par exemple, ceux de Kearfott [56] ou de Domes & Neumaier [29]. De fait, de nombreux travaux ont eu lieu ces dernières années sur des questions connexes comme celle des différentes linéarisations possibles ou encore leurs utilisations. L'ensemble de ces évolutions, alliées avec d'autres apports comme des résultats récents sur la construction de zone d'exclusions [98], sont caractéristiques d'une communauté active dont les résultats sont propres à faire évoluer et à enrichir les techniques de résolution que nous avons proposées.

Les contraintes sur les flottants se caractérisent par un nombre relativement restreint d'acteurs sans doute lié à la difficulté de raisonner à partir d'une arithmétique assez pauvre, mais aussi à l'émergence assez récente du besoin, les calculs en virgule flottante n'ayant fait leur apparition dans les systèmes critiques que depuis quelques années. Pourtant, le caractère de plus en plus critique de ces applications et l'utilisation de plus en plus fréquente de ces calculs dans des systèmes cyber-physiques renforcent la nécessité de pouvoir raisonner sur l'arithmétique des flottants et ses conséquences. Différentes approches commencent à émerger qui vont de l'introduction des flottants dans les solveurs SMT [15, 45] aux approches formelles [11], en passant par l'interprétation abstraite [42, 78, 43, 16]. Cependant, les contraintes sur les flottants présentent un compromis unique entre souplesse d'utilisation, capacités de résolution, d'instanciation et efficacité. Le potentiel est donc là, mais le passage à l'échelle reste une difficulté importante pour les contraintes

sur les flottants. Et c'est sans doute sur ce dernier point que les recherches devront se concentrer pour assurer le succès de cette technique.

La vérification d'un programme sur les flottants passe aussi par l'évaluation des déviations des calculs, i.e., de l'écart entre le calcul sur les flottants et le calcul idéalisé sur les réels. Cette information souligne la conformité ou la non-conformité d'un programme sur les flottants vis-à-vis de sa spécification sur une dimension spécifique au calcul sur les flottants. C'est l'abstraction de cette dimension là qui constitue la principale originalité de Fluctuat en proposant à l'utilisateur un encadrement des erreurs de calcul. Cet encadrement, s'il est capable de détailler les origines de l'erreur, n'en reste pas moins une surestimation de la déviation du calcul. Dans le même domaine, une autre approche bien connue est celle de la méthode stochastique CESTAC [90] qui permet d'évaluer la stabilité d'un programme. Ce n'est cependant pas une méthode complète et si elle est capable de détecter des cas d'instabilité, i.e., des cas où le calcul sur les flottants dérive de manière substantielle de celui sur les réels, l'absence de détection d'un tel cas ne constitue par vraiment une preuve de stabilité. Les techniques que nous avons développées pour résoudre des contraintes sur les flottants permettent d'envisager une autre approche. En particulier, la capacité à plonger ces contraintes dans un problème sur les réels offre la possibilité d'envisager une nouvelle manière d'aborder cette problématique en recherchant l'écart maximal entre flottants et réels. En approximant les contraintes sur les flottants par des systèmes d'équations et inéquations linéaires sur les réels, on obtient un système linéaire capable de capturer l'ensemble des types, i.e. entiers, réels et flottants, qui peuvent intervenir dans cette maximisation, au sein d'un même système. Une approche du même type que celle que nous avons développée pour optimiser des problèmes sur les réels devrait permettre de résoudre cette maximisation de l'écart du calcul et d'offrir une instance capable d'exercer le cas en question. C'est un des axes de recherches que je souhaite développer dans les années à venir.

Les phénomènes de convergences lentes liées aux contraintes ternaires sur les flottants grèvent les performances des solveurs sur les flottants. Si les travaux que j'ai effectués avec Bruno Marre en limitent les effets les plus flagrants, les fonctions de projection inverse ne garantissent pas une réduction optimale pour des fonctions multivariées. Une fonction sur les flottants peut être vue comme la composition d'une fonction sur les réels avec un arrondi, arrondi qui fait correspondre à un sous-ensemble des réels un seul et unique flottant. C'est lors de cette opération qu'une perte d'information a lieu. Des fonctions de projection optimales, i.e., qui calculeraient des domaines dont les bornes seraient solutions de la contrainte, amélioreraient notablement l'efficacité du filtrage. Si ce comportement semble acquis pour les fonctions monovariées, un travail conséquent reste à faire pour obtenir la même propriété pour les fonctions multivariées.

La recherche de solutions d'un système de contraintes sur les flottants

reste un domaine de recherche qui a fait l'objet de peu de travaux. Il y là pourtant de nombreuses pistes à explorer pour améliorer cet aspect des CSP sur les flottants, en particulier, si, comme dans le cas des solveurs SMT, on s'intéresse soit à la production d'une solution ou à la démonstration de l'absence de solution. L'une des pistes envisagées est l'exploitation de la répartition des solutions en tenant compte de la répartition particulière des flottants. En effet, d'une part, plus on se rapproche de zéro, plus la densité des nombres flottants est importante et, à cause de l'arrondi, plus le zéro d'une fonction est susceptible de correspondre à plusieurs valeurs des opérandes, et, d'autre part, plus la valeur de la dérivée de cette fonction est proche de zéro, plus le nombre de flottants correspondants à une valeur donnée de y est important. En tenant compte de ces effets combinés, la recherche peut être orientée vers des zones plus propices à la découverte de solutions. Une autre piste à explorer est l'exploitation de la structure du problème, i.e., un programme. Le comportement d'un programme est entièrement déterminé par ses variables d'entrée. Il suffit donc de se contenter d'énumérer sur ces variables là pour déterminer l'ensemble des variables du problème. Ce procédé reporte néanmoins une bonne part des difficultés sur le filtrage : pour atteindre un point précis du programme, il faut être capable de propager les informations induites d'une instanciation partielle des variables d'entrée vers les variables qui n'ont pas encore été affectées. C'est là où ce type d'approche peut vite rencontrer ses limites. Cette question de la recherche de solution dans des problèmes sur les flottants nous a paru assez importante pour faire l'objet de la thèse d'Heytem Zitoun que je co-encadre.

La coopération entre interprétation abstraite et programmation par contraintes ouvre de nombreuses perspectives. Il y a là un potentiel important pour obtenir des méthodes d'analyse de programmes aux capacités accrues. Bien que préliminaires, nos travaux ont montré tout l'intérêt qu'il y a à faire coopérer ces deux techniques et soulignés leurs complémentarités. Mais des marges de progressions importantes subsistent. Les travaux de Denmat [27] sur l'utilisation de l'interprétation abstraite pour le traitement des boucles dans un solveur de contraintes montrent qu'une coopération plus fine est possible et, sans doute, souhaitable. Les modalités de cette coopération restent à établir, la finesse de la coopération et la finalité de cette coopération pouvant fortement influer sur les choix à opérer. Du point de vue de la programmation par contraintes, l'interprétation abstraite peut être vue comme un processus de filtrage complet, comme dans le cas de nos travaux où l'ensemble du programme est soumis à interprétation abstraite, ou partiel, comme dans le cas des travaux de Denmat, où son utilisation est restreinte à une sous-structure particulière du programme. Son intégration dans le contexte de la programmation par contraintes semble donc assez naturelle. Cette direction est aussi appuyée par les travaux de Marie Pelleau & al [86] qui ont adapté le domaine abstrait des octogones pour un solveur sur les réels. Néanmoins, la direction inverse, l'intégration de techniques issues de la programmation

par contraintes dans les interprètes abstraits ne doit pas être exclue. On peut alors envisager de regarder les CSP comme un moyen d'affiner les résultats de l'interprétation abstraite pour réduire les ambiguïtés liées aux fausses alarmes en s'appuyant sur leur capacité à filtrer finement les domaines et sur cette autre capacité des solveurs de contraintes à fournir des instances, instances qui concrétisent l'existence d'une erreur dans le programme. C'est sur ces problèmes que nous comptons travailler avec l'ensemble des partenaires de l'ANR COVERIF dans les années à venir.

Références

[1] Heikel Batnini : *Contraintes Globales et Heuristiques de Recherche pour les CSPs Continus.* Thèse de doctorat, Université de Nice-Sophia Antipolis, 1 décembre 2005.

[2] Heikel Batnini, Claude Michel et Michel Rueher : Mind the gaps : A new splitting strategy for consistency techniques. *In Proceedings of 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, LNCS 3709, pages 77–91, Sitges, Spain, 1–5 octobre 2005.

[3] Heikel Batnini, Claude Michel et Michel Rueher : Une nouvelle stratégie de recherche pour les csps continus. *In Premières Journées Francophones de Programmation par Contraintes (JFPC'05)*, Lens - Université d'Artois, 8–10 juin 2005.

[4] Mokhtar S. Bazaraa, Hanif D. Sherali et Marakada Chitharanjan Shetty : *Nonlinear Programming : Theory and Algorithms.* John Wiley & Sons, 1993.

[5] Mohammed Said Belaid : *Résolution de contraintes sur les flottants dédiée à la vérification de programmes.* Thèse de doctorat, Université de Nice-Sophia Antipolis, 4 décembre 2013.

[6] Mohammed Said Belaid, Claude Michel et Michel Rueher : Approximating floating-point operations to verify numerical programs. *In 14th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'10)*, ENS Lyon, France, 27–30 septembre 2010.

[7] Mohammed Said Belaid, Claude Michel et Michel Rueher : Boosting local consistency algorithms over floating-point numbers. *In Proceedings of the 18th international conference on Principles and practice of constraint programming (CP'12)*, LNCS 7514, pages 127–140, Québec City, QC, Canada, 8–12 octobre 2012.

[8] Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers et Jean-François Puget : Revising hull and box consistency. *In INT.*

CONF. ON LOGIC PROGRAMMING, pages 230–244. MIT press, 1999.

[9] Frédéric Benhamou, Frédéric Goualard, Éric Languénou et Marc Christie : Interval constraint solving for camera control and motion planning. *ACM Transactions on Computational Logic (TOCL)*, pages 732–767, octobre 2004.

[10] Fédéric Benhamou, David McAllester et Pascal Van-Hentenryck : Clp(intervals) revisited. *In Proceedings of the International Symposium on Logic Programming*, pages 124–138, 1994.

[11] Sylvie Boldo et Jean-Christophe Filliâtre : Formal verification of floating-point programs. *In 18th IEEE Symposium on Computer Arithmetic (ARITH-18 2007)*, pages 187–194, Montpellier, France, 25–27 juin 2007.

[12] Glencora Borradaile et Pascal Van Hentenryck : Safe and tight linear estimators for global optimization. *Mathematical Programming*, 102(3):495–517, 2005.

[13] Bernard Botella, Arnaud Gotlieb et Claude Michel : Symbolic execution of floating-point computations. *Software Testing, Verification and Reliability*, 16(2):97–121, 2006.

[14] Bernard Botella, Arnaud Gotlieb, Claude Michel, Michel Rueher et Patrick Taillibert : Utilisation des contraintes pour la génération automatique de cas de test structurels. *Techniques et science informatiques (TSI)*, 21(9):1163–1187, 2002.

[15] Angelo Brillout, Daniel Kroening et Thomas Wahl : Mixed abstractions for floating-point arithmetic. *In Proceedings of Formal Methods in Computer-Aided Design (FMCAD) 2009*, pages 69–76, Austin, Texax, 15–18 novembre 2009. IEEE.

[16] Liqian Chen, Antoine Miné et Patrick Cousot : A sound floating-point polyhedra abstract domain. *In Proc. of the Sixth Asian Symposium on Programming Languages and Systems (APLAS'08)*, volume 5356 de *Lecture Notes in Computer Science (LNCS)*, pages 3–18. Springer, décembre 2008.

[17] Edmund Clarke, Daniel Kroening et Flavio Lerda : A tool for checking ANSI-C programs. *In TACAS*, volume 2988 de *LNCS*, pages 168–176, 2004.

[18] Hélène Collavizza, François Delobel et Michel Rueher : A note on partial consistencies over continuous domains solving techniques. *In Proceedings of CP98 (Fourth International Conference on Principles and Practice of Constraint Programming)*, Pisa, Italy, 26–30 octobre 1998.

[19] Hélène Collavizza, François Delobel et Michel Rueher : Extending consistent domains of numeric csp. *In Proceedings of the 16th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'99, pages 406–411, 1999.

[20] Hélène Collavizza, Claude Michel, Olivier Ponsini et Michel Rueher : Generating test cases inside suspicious intervals for floating-point number program. *In 6th Workshop on Constraints in Software Testing, Verification, and Analysis*, Hyderabad, India, 31 mai 2014.

[21] Hélène Collavizza, Michel Rueher et Pascal Van Hentenryck : Cpbpv : A constraint-programming framework for bounded program verification. *Constraints*, 15(2):238–264, 2010.

[22] Roberto Di Cosmo, Olivier Lhomme et Claude Michel : Aligning component upgrades. *In Second Workshop on Logics for Component Configuration (LoCoCo'11)*, volume 65 de *EPTCS*, pages 1–11, Perugia, Italy, 12 septembre 2011.

[23] Patrick Cousot et 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 ACM Symposium on Principles of Programming Languages*, pages 238–252, jan 1977.

[24] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné et Xavier Rival : Why does astrée scale up ? *Formal Methods in System Design*, 35(3):229–264, 2009.

[25] David Delmas, Eric Goubault, Sylvie Putot, Jean Souyris, Karim Tekkal et Franck Védrine : Towards an industrial use of FLUCTUAT on safety-critical avionics software. *In FMICS*, volume 5825 de *LNCS*, pages 53–69. Springer, 2009.

[26] Giorgio Delzanno et Andreas Podelski : Model checking in CLP. *In Tools and Algorithms for the Construction and Analysis of Systems, 5th International Conference, TACAS '99*, Amsterdam, The Netherlands, 22–28 mars 1999.

[27] Tristan Denmat, Arnaud Gotlieb et Mireille Ducassé : An abstract interpretation based combinator for modelling while loops in constraint programming. *In 13th International Conference on Principles and Practice of Constraint Programming*, pages 241–255, Providence, Rhode Island, 23–27 septembre 2007.

[28] Peter Dietmaier : The stewart-gough platform of general geometry can have 40 real postures. *In Advances in Robot Kinematics : Analysis and Control*, pages 1–10, 1998.

[29] Ferenc Domes et Arnold Neumaier : Rigorous filtering using linear relaxations. *Journal of Global Optimization*, 53(3):441–473, juillet 2012.

[30] Peter Dorato, Kun Li, E. B. Kosmatopoulos, P. A. Ioannou et H. Ryaciotaki-Boussalis : Quantified multivariate polynomial inequalities. the mathematics of practical control design problems. *IEEE Control Systems Magazine*, (5):48–58, 2000.

[31] Peter Dorato, Wei Yang et Chaouki Abdallah : Robust multiobjective feedback design by quantifier elimination. *Journal of Symbolic Computation*, 24(2):153–159, 1997.

[32] Vijay D'Silva, Leopold Haller, Daniel Kroening et Michael Tautschnig : Numeric bounds analysis with conflict-driven learning. *In Proc. TACAS*, volume 7214 de *Lecture Notes in Computer Science*, pages 48–63. Springer, 2012.

[33] G. Fiorio, Stefano Alberto Malan, Mario Milanese et Michele Taragna : Robust performance design of fixed structure controllers for systems with uncertain parameters. *In Proceedings of the 32nd IEEE Conference on Decision and Control*, volume 4, pages 3029–3031, San Antonio, Texas, 15–17 décembre 1993.

[34] Andreas Frommer et Bruno Lang : Existence tests for solutions of nonlinear equations using borsuk's theorem. *SIAM Journal on Numerical Analysis*, 43(3):1348—-1361, juillet 2006.

[35] Alexandre Goldsztejn : A branch and prune algorithm for the approximation of non-linear ae-solution sets. *In SAC'06 : Proceedings of the 2006 ACM symposium on Applied computing*, pages 1650–1654, Dijon, France, 23–27 avril 2006.

[36] Alexandre Goldsztejn et Luc Jaulin : Inner and outer approximations of existentially quantified equality constraints. *In Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming, (CP 2006)*, pages 198–212, Nantes, France, 25–29 septembre 2006.

[37] Alexandre Goldsztejn, Yahia Lebbah, Claude Michel et Michel Rueher : Revisiting the upper bounding process in a safe branch and bound algorithm. *In Proceedings of the 14th international conference on Principles and Practice of Constraint Programming (CP'08)*, LNCS 5202, pages 598–602, Sydney, Australia, 14–18 septembre 2008.

[38] Alexandre Goldsztejn, Claude Michel et Michel Rueher : An efficient algorithm for a sharp approximation of universally quantified inequalities. *In Proceedings of the 2008 ACM symposium on Applied computing (SAC'08)*, pages 134–139, Fortaleza, Ceara, Brazil, 16–20 mars 2008.

[39] Alexandre Goldsztejn, Claude Michel et Michel Rueher : Efficient handling of universally quantified inequalities. *Constraints*, 14(1):117–135, 2009.

[40] Arnaud Gotlieb, Bernard Botella et Michel Rueher : Automatic Test Data Generation Using Constraint Solving Techniques. *In Proceedings of ISSTA'98*, pages 53–62, Clearwater Beach, FL, mars 1998.

[41] Arnaud Gotlieb, Bernard Botella et Michel Rueher : A CLP framework for computing structural test data. *In Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings*, pages 399–413, London, UK, 24–28 juin 2000.

[42] Eric Goubault : Static analyses of the precision of floating-point operations. *In Static Analysis, 8th International Symposium*, pages 234–259, juillet 2001.

[43] Eric Goubault et Sylvie Putot : Static analysis of numerical algorithms. *In SAS*, volume 4134 de *LNCS*, pages 18–34. Springer, 2006.

[44] Eric Goubault et Sylvie Putot : Robustness analysis of finite precision implementations. *In Programming Languages and Systems - 11th Asian Symposium, APLAS*, volume 8301 de *LNCS*, pages 50–57. Springer, 2013.

[45] Leopold Haller, Alberto Griggio, Martin Brain et Daniel Kroening : Deciding floating-point logic with systematic abstraction. *In Proceedings of Formal Methods in Computer-Aided Design (FMCAD) 2012*, pages 131–140, Cambridge, 22–25 octobre 2012.

[46] Eldon Hansen et R. Greenberg : An interval newton method. *Applied Mathematics and Computations*, pages 12(1983)89—98, 1983.

[47] Eldon Hansen et G. William Walster : *Global Optimization Using Interval Analysis : Revised and Expanded*. Marcel Dekker, New York, 2003.

[48] Eldon Hansen et G. William Walster : *Global Optimization Using Interval Analysis*. Marcel Dekker, 2004.

[49] Timothy J. Hickey, Qun Ju et Maarten H. van Emden : Interval arithmetic : From principles to implementation. *Journal of ACM*, 48(5):1038–1068, septembre 2001.

[50] Siriporn Hongthong et R. Baker Kearfott : Rigorous linear overestimators and underestimators. Rapport technique, university of Lousiana, 2004.

[51] Rainer Horst et Hoang Tuy : *Global Optimization : Deterministic Approches*. Springer-Verlag, 1993.

[52] Luc Jaulin et Éric Walter : Guaranteed tuning, with application to robust control and motion planning. *Automatica*, pages 1217–1221, août 1996.

[53] Mats Jirstrand : Nonlinear control system design by quantifier elimination. *Journal of Symbolic Computation - Special issue : applications of quantifier elimination archive*, pages 137–152, août 1997.

[54] R. Baker Kearfott : *Rigorous Global Search : Continuous Problems.* Kluwer Academic Publishers Group, 1996.

[55] R. Baker Kearfott : Validated probing with linear relaxations. *submitted to Journal of Global Optimization*, 2005.

[56] R. Baker Kearfott : Discussion and empirical comparisons of linear relaxations and alternate techniques in validated deterministic global optimization. *journal of Optimization Methods and Software*, pages 715–731, octobre 2006.

[57] Yahia Lebbah, Claude Michel et Michel Rueher : Efficient pruning technique based on linear relaxations. *In Second international conference on Global Optimization and Constraint Satisfaction (COCOS'03)*, Lausanne, Switzerland, 18–21 novembre 2003.

[58] Yahia Lebbah, Claude Michel et Michel Rueher : An efficient and safe framework for solving optimisation problems. *In 11th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics (SCAN'04)*, Fukuoka, Japan, 4–8 octobre 2004.

[59] Yahia Lebbah, Claude Michel et Michel Rueher : Efficient pruning technique based on linear relaxations. *In Global Optimization and Constraint Satisfaction (selected papers from COCOS'03)*, LNCS 3478, pages 1–14, 2005.

[60] Yahia Lebbah, Claude Michel et Michel Rueher : A rigorous global filtering algorithm for quadratic constraints. *Constraints*, 10(1):47–65, 2005.

[61] Yahia Lebbah, Claude Michel et Michel Rueher : An efficient and safe framework for solving optimization problems. *Journal of Computational and Applied Mathematics*, 199(2):372–377, 2007.

[62] Yahia Lebbah, Claude Michel et Michel Rueher : Using constraint techniques for a safe and fast implementation of optimality-based reduction. *In Proceedings of the 2007 ACM symposium on Applied computing (SAC'07)*, pages 326–331, Seoul, Korea, 11–15 mars 2007.

[63] Yahia Lebbah, Claude Michel et Michel Rueher : Utilisation des techniques de programmation par contraintes pour une implémentation rigoureuse et efficace de la réduction basée sur l'optimalité. *In Troisièmes Journées Francophones de Programmation par Contraintes, JFPC 2007*, Rocquencourt, France, 4–6 juin 2007.

[64] Yahia Lebbah, Claude Michel, Michel Rueher, David Daney et Jean-Pierre Merlet : Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2005.

[65] Yahia Lebbah, Michel Rueher et Claude Michel : A global filtering algorithm for handling systems of quadratic equations and inequations. *In Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, LNCS 2470, pages 109–123, Cornell University, Ithaca, NY, USA, 7–13 septembre 2002.

[66] Olivier Lhomme : Consistency techniques for numeric csps. *In Proceedings of IJCAI'93*, pages 232–238, 1993.

[67] Leo Liberti et Constantinos C. Pantelides : Convex envelopes of monomials of odd degree. *J. Global Optimization*, 25(2):157–168, 2003.

[68] Alan K. Mackworth : Consistency in networks of relations. *Journal of Artificial Intelligence*, pages 8(1) :99–118, 1977.

[69] Stefano Alberto Malan, Mario Milanese et Michele Taragna : Robust analysis and design of control systems using interval arithmetic. *Automatica*, 33(7):1363–1372, juillet 1997.

[70] Bruno Marre et Claude Michel : Improving the floating point addition and subtraction constraints. *In Proceedings of the 16th international conference on Principles and practice of constraint programming (CP'10)*, LNCS 6308, pages 360–367, St. Andrews, Scotland, 6–10 septembre 2010.

[71] Garth P. McCormick : Computability of global solutions to factorable nonconvex programs – part i – convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[72] Claude Michel : *Modèles et Implémentations d'Interprètes Réflexifs*. Thèse de doctorat, Université de Nice-Sophia Antipolis, 5 décembre 1997.

[73] Claude Michel : Exact projection functions for floating point number constraints. *In AI&M 1-2002, Seventh international symposium on Artificial Intelligence and Mathematics (7th ISAIM)*, Fort Lauderdale, Floride (US), 2–4 janvier 2002.

[74] Claude Michel, Mohammed Said Belaid, Olivier Ponsini et Michel Rueher : An hybrid approach to solve constraints over the floating-point numbers. *In 13th INFORMS Computing Society Conference (ICS)*, Santa Fe, New Mexico, USA, 6–8 janvier 2013.

[75] Claude Michel, Yahia Lebbah et Michel Rueher : Safe embedding of the simplex algorithm in a csp framework. *In 5th Int. Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR'03)*, pages 210–220, CRT, Université de Montréal, 8–10 mai 2003.

[76] Claude Michel et Michel Rueher : Handling software upgradeability problems with milp solvers. *In First Workshop on Logics for Component Configuration (LoCoCo'10)*, volume 29 de *EPTCS*, pages 1–10, Edinburg, Scotland, 10 juillet 2010.

[77] Claude Michel, Michel Rueher et Yahia Lebbah : Solving constraints over floating-point numbers. *In Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP'01)*, LNCS 2239, pages 524–538, Paphos, Chypre, novembre 2001.

[78] Antoine Miné : *Weakly relational numerical abstract domains.* Thèse de doctorat, École Polytechnique, décembre 2004.

[79] Michel Minoux : *Mathematical Programming. Theory, Algorithms and Applications.* Wiley, 1986.

[80] Antoine Miné : Relational abstract domains for the detection of floating-point run-time errors. *In Proc. of the European Symposium on Programming (ESOP'04)*, volume 2986 de *Lecture Notes in Computer Science (LNCS)*, pages 3–17. Springer, mars 2004.

[81] Bruce A. Murtagh et Michael A. Saunders : Minos 5.5 user's guide. Rapport technique SOL 83-20R, Systems Optimization Laboratory, Dep. of Operations Research, Stanford University, juillet 1998.

[82] Arnold Neumaier : Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, 2004.

[83] Arnold Neumaier et Oleg Shcherbina : Safe bounds in linear and mixed-integer programming. *Mathematical Programming*, pages 283–296, 2004.

[84] Bertrand Neveu, Gilles Trombettoni et Ignacio Araya : Node selection strategies in interval branch and bound algorithms. *J. Global Optimization*, 64(2):289–304, 2016.

[85] James M. Ortega et Werner C. Rheinboldt : *Iterative Solution of Nonlinear Equations in Several Variables.* SIAM, 1970.

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

[87] Andreas Podelski : Model checking as constraint solving. *In Static Analysis, 7th International Symposium, SAS 2000*, pages 22–37, Santa Barbara, CA, USA, juin 2000.

[88] Olivier Ponsini, Claude Michel et Michel Rueher : Refining abstract interpretation based value analysis with constraint programming techniques. *In Proceedings of the 18th international conference on Principles and practice of constraint programming (CP'12)*, LNCS 7514, pages 593–607, Québec City, QC, Canada, 8–12 octobre 2012.

[89] Olivier Ponsini, Claude Michel et Michel Rueher : Verifying floating-point programs with constraint programming and abstract interpretation techniques. *Automated Software Engineering*, 23(2):191–217, juin 2016.

[90] Michel La Porte et Jean Vignes : Etude statistique des erreurs dans l'arithmétique des ordinateurs ; application au contrôle des résultats d'algorithmes numériques. *Numerische Mathematik*, 23:63–72, 1974/75.

[91] Jean-François Puget et Pascal Van-Hentenryck : A constraints satisfaction approach to a circuit design problem. *Journal of global optimization*, pages 13(1) :75–93, 1998.

[92] C. Radhakrishna Rao et Sujit Kumar Mitra : *Generalized Inverse of Matrices and Its Applications*. New York : Wiley, 1971.

[93] Stefan Ratschan : Approximate quantified constraint solving by cylindrical box decomposition. *Reliable Computing*, 8(1):21–42, 2002.

[94] Stefan Ratschan : Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, octobre 2006.

[95] Dietmar Ratz : Box-splitting strategies for the interval gauss-seidel step in a global optimization method. *Computing*, 53(3-4):337–353, 1994.

[96] Hong S. Ryoo et Nikolaos V. Sahinidis : A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, mars 1996.

[97] Nikolaos V. Sahinidis et Mohit Tawarmalani : Baron 5.0 : Global optimisation of mixed-integer nonlinear programs. Rapport technique, University of Illinois at Urbana-Champaign, Department of Chemical and Biomolecular Engeneering, 2002.

[98] Hermann Schichl, Mihály Csaba Markót et Arnold Neumaier : Exclusion regions for optimization problems. *Journal of Global Optimization*, 59(2):569–595, juillet 2014.

[99] Hanif D. Sherali et Cihan H. Tuncbilek : New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. *Operations Research Letters*, pages 21 :1–9, 1997.

[100] Naum Zuselevich Shor : Dual quadratic estimates in polynomial and boolean programming. *Annals of Operations Research*, pages 25 :163–168, 1990.

[101] Mohit Tawarmalani et Nikolaos V. Sahinidis, éditeurs. *Convexification and Global Optimization in Continuous and Mixed-Integer Non-Linear Programming*. Kluwer Academic Publishers, 2002.

[102] Charlotte Truchet, Marie Pelleau et Frédéric Benhamou : Abstract domains for constraint programming, with the example of octagons. *In 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2010*, pages 72–79, Timisoara, Romania, 23–26 septembre 2010.

[103] Xuan-Ha Vu, Djamila Sam-Haroud et Marius-Calin Silaghi : Approximation techniques for non-linear problems with continuum of solutions. *In Abstraction, Reformulation and Approximation, 5th International Symposium, SARA 2002*, pages 224–241, Kananaskis, Alberta, Canada, 2–4 août 2002.

[104] Michael Zettler et Jürgen Garloff : Robustness analysis of polynomials with polynomial parameter dependency using Bernstein expansion. *IEEE Tansactions on Automatic Control*, 43(3):425–431, 1998.

Annexe : Table des principales notations

$\mathcal{R}$	l'ensemble des réels
$\mathcal{F}$	un ensemble de flottants
x	une variable de type scalaire ou vecteur, l'ambiguïté étant levée par le texte
x_i	la ième composante du vecteur x
$\mathbf{x}$	un intervalle ou un vecteur d'intervalles, l'ambiguïté étant levée par le texte
$\underline{x}$	la borne inférieure de l'intervalle $\mathbf{x}$
$\overline{x}$	la borne supérieure de l'intervalle $\mathbf{x}$
$[\underline{x}, \overline{x}]$	un intervalle fermé, e.g., $[\underline{x}, \overline{x}] = \{x \in \mathcal{R}, \underline{x} \leq x \leq \overline{x}\}$
$]\underline{x}, \overline{x}[$	un intervalle ouvert, e.g., $]\underline{x}, \overline{x}[= \{x \in \mathcal{R}, \underline{x} < x < \overline{x}\}$
A	une matrice
$\mathbf{A}$	une matrice d'intervalles
x^+	le successeur du flottant x
x^-	le prédécesseur du flottant x