HAL
open science

Contraintes globales et heuristiques de recherche pour les CSPs continus

Heikel Batnini

To cite this version:

Heikel Batnini. Contraintes globales et heuristiques de recherche pour les CSPs continus. Génie logiciel [cs.SE]. Université Nice Sophia Antipolis, 2005. Français. NNT : . tel-00091375

HAL Id: tel-00091375

https://theses.hal.science/tel-00091375v1

Submitted on 6 Sep 2006

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

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

THÈSE

présentée à

L'UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS
U.F.R. SCIENCES
École Doctorale des Sciences et Technologies de l'Information et de la Communication

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité

INFORMATIQUE

par

Heikel BATNINI

dirigée par : **Michel RUEHER**
co-dirigée par : **Claude MICHEL**

Contraintes Globales et Heuristiques de Recherche pour les CSPs Continus

Soutenue publiquement le 1er Décembre 2005 devant le jury composé de

M.	**Frédéric**	**BENHAMOU**	Professeur	Rapporteur
M.	**Patrice**	**BOIZUMAULT**	Professeur	Président
M.	**Claude**	**MICHEL**	Ingénieur de recherche	Co-directeur
M.	**Jean-Charles**	**RÉGIN**	ILOG S.A.	Examinateur
M.	**Michel**	**RUEHER**	Professeur	Directeur
M.	**Pascal**	**VAN HENTENRYCK**	Professeur	Rapporteur

Table des matières

1 Introduction **1**
1.1 Contraintes globales pour les CSPs continus 1
1.2 Techniques de résolution pour les CSPs continus 5

I État de l'art **9**

2 CSPs à domaines finis **11**
2.1 Définition et exemples 11
2.2 Filtrage par consistance 14
2.2.1 Arc-consistance 15
2.2.2 Consistances partielles 17
2.3 Résolution classique de CSPs finis 18
2.4 Amélioration de l'algorithme de recherche 20
2.5 Contraintes globales 21

3 CSPs à domaines continus **25**
3.1 Définition et exemple 25
3.2 Méthodes algébriques et numériques 26
3.3 Arithmétique des intervalles 27
3.3.1 Définitions et notations 27
3.3.2 Opérateurs arithmétiques 28
3.3.3 Extension aux intervalles 29
3.3.4 Problème de dépendance des variables 31
3.4 Résolution de CSPs continus 31
3.4.1 Heuristiques de recherche 32
3.5 Consistances sur les CSPs continus 33
3.5.1 Consistances locales 33
3.5.2 Consistances partielles 38
3.6 Contraintes globales 39
3.7 Autres techniques de résolution 40

II Contraintes globales pour les CSPs continus **43**

4 Contraintes de distance et applications **45**
4.1 Définitions et notations 45

4.2 Modélisations 46
4.3 Applications 46

5 Ajout de contraintes redondantes 51
5.1 Fermeture du graphe de contraintes 51
5.1.1 Calcul de la fermeture 52
5.1.2 Résultats expérimentaux 55
5.2 Introduction de barycentres 55

6 QuadDist : Un filtrage global pour les contraintes de distance 61
6.1 Introduction 61
6.1.1 Présentation informelle 62
6.2 Quad :un filtrage global pour les contraintes quadratiques 64
6.3 QuadDist : Une approche globale 64
6.3.1 Schéma général 64
6.3.2 Décomposition sémantique des domaines 65
6.3.3 Relaxation des contraintes de distance 67
6.3.4 Combinaison des approximations 68
6.4 Taille des programmes linéaires générés 69
6.5 Résultats expérimentaux 70
6.5.1 Problèmes aléatoires 71

III Contribution à la résolution de CSPs continus 73

7 Décomposition sémantique pour les contraintes de distance 75
7.1 Introduction 75
7.2 Description informelle 76
7.2.1 Décomposition sémantique des domaines 76
7.2.2 Décomposition sémantique d'un CSP 78
7.3 Décomposition sémantique d'un CSP 79
7.3.1 Décomposition sémantique d'une contrainte de distance 80
7.3.2 Réécriture du CSP initial 81
7.3.3 Algorithme de décomposition sémantique 81
7.4 Résultats expérimentaux 83
7.4.1 Problème classique du pentagone 83
7.4.2 Cinématique directe d'un robot plan 84
7.4.3 Résultats et analyse 86

8 Mind The Gaps : Une nouvelle stratégie de résolution pour les techniques de consistances 87
8.1 Introduction 87
8.2 MindTheGaps : Schéma général 89
8.3 Consistances locales et trous 90
8.3.1 Extensions aux intervalles et fonctions de projection 91
8.3.2 Hull-consistance et trous 91
8.3.3 Box-consistance et trous 94

8.4 Résultats expérimentaux 95
8.4.1 MindTheGaps : heuristiques 96
8.4.2 Analyse des résultats 97
8.4.3 Évaluation de contraintes et trous 99
8.5 Extension aux consistances partielles 100

9 Conclusion **103**

Annexe A **107**

Table des figures

1.1 Un manipulateur planaire de type 3-RPR 3
1.2 Fermeture d'un graphe de contraintes de distance 4
1.3 Exemple de linéarisation par QuadDist. 5
1.4 Décomposition sémantique du problème du pentagone. 6

2.1 Graphe de contraintes associé au CSP de l'exemple 2.1.1 page 12 13
2.2 Micro-structure associée au CSP de l'exemple 2.1.1 page 12 14
2.3 Arc-consistance pour un problème de coloriage de graphe 16
2.4 Procédure Revise 16
2.5 Algorithme de propagation AC-3 [84] 17
2.6 Filtrage par arc-consistance sur le CSP de l'exemple 2.1.1 17
2.7 Algorithme de backtrack sur le CSP de l'exemple 2.1.1 page 12 20
2.8 Illustration d'un filtrage par la contraint globale AllDiff 23

3.1 Schéma général de l'algorithme de recherche standard 32
3.2 Algorithme de propagation IN-1 34
3.3 Projection d'une contrainte 35
3.4 Mise en œuvre de HC4revise sur la contrainte $y = x^2$ avec $(x, y) \in [-2, 4] \times [1, 16]$. 36
3.5 L'opérateur de Narrowing associé à la Box-consistance 37
3.6 LeftNarrow :L'opérateur de réduction de la borne inférieure, qui recherche le quasi-zéro le plus à droite. 37
3.7 3BPrune : Algorithme de fermeture par 3B-consistance 38
3.8 Décomposition en 2^k-arbre de la contrainte $x^2+y^2 <= 1$, avec $\boldsymbol{x} = \boldsymbol{y} = [-1, 1]$: les pavés Inner sont représentées en blanc, les zones Outer en gris foncé et les zone Union en gris clair. 41

5.1 Fermeture du graphe de contraintes représentant le DCSP de l'exemple 5.1.2 53
5.2 Algorithme BSFilter qui calcule la fermeture du graphe de contraintes 54
5.3 Problème des ponts[59] 56
5.4 Le problème des losanges [59] 57
5.5 Description du CSP T_{ijk} 58
5.6 Description du CSP T'_{ijk} 59
5.7 Comparaison des résultats pour la 2B-consistance, la 3B-consistance et la 3B-consistance sur tous les triangles avec introduction du barycentre. 59

6.1 Réduction du domaine par 2B-consistance 62
6.2 Première approximation des solutions de $\mathcal{L}(\mathcal{C})$. 63

6.3 Illustration de la décomposition sémantique sur l'exemple 7.2.1. 66
6.4 Relaxation de c sur $\mathcal{P}^{++}$. 67
6.5 Sous-estimations de trois sous-domaines. 69
6.6 Comparaison des temps de calculs pour Quad et pour QuadDist 71
6.7 Temps d'execution par rapport au nombre de variables des systèmes linéaires engendrés 72

7.1 Disjonctions dans l'espace des solutions 75
7.2 La figure de gauche montre les domaines de A et B après un filtrage par 2B-consistance. Les carrés pleins représentent les continuums de solutions. La figure de droite montre les 4 sous-domaines de A et ceux de B ainsi que leurs liaisons dans la micro-structure calculée par la SDD. 77
7.3 La figure 7.3(a) montre la décomposition du domaine de B en 2 sous-domaines B_1 et B_2, pour la contrainte c_1. La figure 7.3(b) montre la décomposition du domaine de C en 4 sous-domaines C_1, C_2, C_3 et C_4, pour la contrainte c_2. . . 78
7.4 Les 3 solutions du CSP de l'exemple 7.2.2 sont les triangles ABC, $AB'C'$ et $AB''C''$. 79
7.5 Solutions du problème du pentagone 83
7.6 Ajout de contraintes dans le problème du pentagone 84
7.7 Un manipulateur planaire de type 3-RPR 85
7.8 Résultats expérimentaux 85

8.1 Schéma général de MindTheGaps. 90
8.2 HCNarrow : l'opérateur de réduction de la Hull-consistance 92
8.3 HCNarrow* : extension de HCNarrow qui collecte les trous 93
8.4 HCPrune* enforces Hull-consistency and collects the gaps 93
8.5 Opérateur de réduction de la Box-consistance étendu pour collecter les trous 95

1 Un triangle ABC et son centre de gravité G 107
2 Introduction du centre de gravité dans le triangle $\mathcal{P}_i\mathcal{P}_j\mathcal{P}_k$ 109

Liste des tableaux

5.1 Comparaison des filtrages avec et sans fermeture du graphe de contraintes . . 53
5.2 Résultats des différents filtrages pour le problème des ponts 56
5.3 Résultats des différents filtrages pour le problème des losanges 57
5.4 Réduction des domaines du CSP de l'exemple 5.1.2 60

6.1 Réduction des domaines et nombre d'itérations de Quad en comparaison avec QuadDist 71

8.1 Résultats expérimentaux pour HC4 (à gauche) et HC4+Newton (à droite). . . 98
8.2 Résultat expérimentaux pour Box (à gauche) et avec le critère de Hansen (à droite) 98
8.3 Résultat expérimentaux pour Box+Newton (à gauche) et avec le critère de Hansen (à droite) 99
8.4 Résultats expérimentaux pour *ecoN* et les formes de Horner corresponantes *ecoNH*. 100
8.5 Résultats expérimentaux de MindTheGaps pour la 3B-consistance 101

Glossaire des notations

$\lvert E\rvert$	Le cardinal de l'ensemble E.
$(\mathcal{X}, \mathcal{D}, \mathcal{C})$	un CSP fini.
$\mathcal{X} = \{x_1, \ldots, x_n\}$	L'ensemble des variables.
$\mathcal{D} = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n\}$	L'ensemble des domaines finis.
$\mathcal{C} = \{c_1, \ldots, c_m\}$	L'ensemble des contraintes.
n	le nombre de variables de $\mathcal{X}$ ($n = \lvert\mathcal{X}\rvert$).
$\boldsymbol{d}_i$	le domaine fini associé à la variable x_i.
d	le nombre d'éléments du plus grand domaine de $\mathcal{D}$.
v_i	un élément du domaine fini $\boldsymbol{d}_i$.
m	le nombre de contraintes de $\mathcal{C}$ ($m = \lvert\mathcal{C}\rvert$).
$\mathcal{X}(c)$	l'ensemble des variables de la contrainte c.
$(v_1, \ldots, v_n) \in c$	c est satisfaite lorsque $x_1 = v_1, \ldots, x_n = v_n$.
$(v_1, \ldots, v_n) \notin c$	c n'est est pas satisfaite lorsque $x_1 = v_1, \ldots, x_n = v_n$.
$\mathbb{R}$	L'ensemble des nombre réels.
$\overline{\mathbb{R}}$	$\mathbb{R}$ étendu à $\{-\infty, +\infty\}$.
$\mathbb{F}$	L'ensemble des nombre flottants dans un format donné ($\overline{\mathbb{F}} \subset \overline{\mathbb{R}}$).
$\overline{\mathbb{F}}$	$\mathbb{F}$ étendu à $\{-\infty, +\infty\}$.
$[\underline{x}, \overline{x}]$	un intervalle fermé (l'ensemble des valeurs réelles x telles que $\underline{x} \leq x \leq \overline{x}$).
$]\underline{x}, \overline{x}[$	un intervalle ouvert (l'ensemble des valeurs réelles x telles que $\underline{x} < x < \overline{x}$).
$m(\boldsymbol{x})$	le milieu de l'intervalle $\boldsymbol{x}$ ($m(\boldsymbol{x}) = (\overline{x} + \underline{x})/2$).
$w(\boldsymbol{x})$	la taille de l'intervalle $\boldsymbol{x}$ ($w(\boldsymbol{x}) = \overline{x} - \underline{x}$).
$\mathbb{IR}$	l'ensemble des intervalles à bornes dans $\overline{\mathbb{R}}$.
$\mathbb{IF}, \mathbb{I}$	l'ensemble des intervalles à bornes dans $\overline{\mathbb{F}}$.
$\cap_{\mathbb{I}}$	l'opérateur d'intersection sur $\mathbb{I}$
x, y, z	des variables réelles.
$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$	des intervalles.
X, Y, Z	des vecteurs de variables réelles.
$\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$	des vecteurs d'intervalles - ou *boîtes*.
$\boldsymbol{X} = \emptyset$	si une des composantes $\boldsymbol{x}_i$ de $\boldsymbol{X}$ est vide
$w(\boldsymbol{X})$	La taille du plus grand intervalle qui compose $\boldsymbol{X}$
$\mathcal{P}_i(x_{i_1}, \ldots, x_{i_p})$	Un point de $\mathbb{R}^p$
$\mathcal{P}_i(x_i, y_i, z_i)$	Un point de $\mathbb{R}^3$
$\mathcal{P}_i(x_i, y_i)$	Un point de $\mathbb{R}^2$
$\overrightarrow{\mathcal{P}_i\mathcal{P}_j}$	le vecteur entre le point $\mathcal{P}_i$ et le point $\mathcal{P}_j$
δ_{ij}	la distance entre le point $\mathcal{P}_i$ et le point $\mathcal{P}_j$
$\boldsymbol{\mathcal{P}}_i$	le domaine du point $\mathcal{P}_i$, c'est-à-dire, $\boldsymbol{\mathcal{P}}_i = \boldsymbol{x}_{i_1} \times \ldots \times \boldsymbol{x}_{i_p}$
$\boldsymbol{u} = \bigcup \boldsymbol{u}_{(j)}$	une union d'intervalles
$\lvert\boldsymbol{u}\rvert$	le nombre de sous-intervalles qui composent $\boldsymbol{u}$
$\underline{\boldsymbol{u}}$	la borne inférieure de $\boldsymbol{u}$
$\overline{\boldsymbol{u}}$	la borne supérieure de $\boldsymbol{u}$
$\mathbb{U}$	l'ensemble des unions d'intervalles sur à bornes dans $\mathbb{F}$
$\boldsymbol{U}, \boldsymbol{V}$	des vecteurs d'unions d'intervalles
$\cap_{\mathbb{U}}$	l'opérateur d'intersection sur $\mathbb{U}$

Chapitre 1

Introduction

Beaucoup d'applications allant de la robotique à la chimie et la géométrie peuvent être exprimées comme des problèmes de satisfaction de contraintes numériques ou NCSP. Un NCSP (Numerical Constraint Satisfaction Problem) est défini par un ensemble de variables et un ensemble de contraintes non-linéaires sur ces variables. Le domaine de variation de ces variables sont des intervalles fermés sur $\mathbb{R}$, autrement dit des domaines continus. Les CSPs numériques, ou CSPs continus, peuvent exprimer une large classe de problème, en particulier des problèmes avec des données imprécises ou des paramètres à valeurs bornées. Le but est de trouver des boîtes qui approximent au mieux les solutions de ces contraintes.

En général, les méthodes algébriques sont incapables de traiter ces problèmes et les méthodes numériques ne garantissent pas la correction des résultats. Toutefois, des approximations correctes des solutions peuvent être obtenues par des solveurs de contraintes basés sur l'arithmétique des intervalles. La plupart de ces solveurs implémentent un algorithme de recherche qui combine une technique d'énumération de l'espace de recherche, des techniques de consistances locales, mais également des techniques issues de l'analyse numérique.

Les consistances locales pour des CSPs sur les domaines finis ont été adaptées aux CSPs numériques. Les algorithmes de filtrage associés éliminent des domaines certaines valeurs inconsistantes, c'est-à-dire pour lesquelles il existe au moins une contrainte qui n'est pas vérifiée. En pratique, le filtrage est en général limité à la contraction des bornes des intervalles.

Les techniques classiques pour résoudre des CSPs numériques sont basées sur un algorithme de type *Branch&Prune*. Cet algorithme alterne filtrages et décompositions de domaines jusqu'à ce qu'une précision donnée sur les domaines soit atteinte. L'étape de décomposition (ou *splitting*) sélectionne une variable et découpe son domaine en plusieurs parties. Les sous-problèmes générés sont alors résolus de manière indépendante.

1.1 Contraintes globales pour les CSPs continus

L'un des inconvénients des consistances locales est que ce sont des méthodes générales, qui sont incapables de tirer parti des spécificités de certains types de problèmes. Il est apparu nécessaire d'introduire des techniques de filtrage adaptées à certains types de contraintes : les *contraintes globales*. De manière générale, une contrainte globale est définie par la conjonction d'un ensemble de contraintes. La prise en compte de la simultanéité de ces contraintes permet l'utilisation d'algorithmes de filtrage plus puissants que les algorithmes de filtrage par

consistance classiques.

La notion de contrainte globale est très largement répandue dans la communauté des CSPs aux domaines finis (par exemple [12, 104, 105]). De nombreuses contraintes globales ont été définies durant ces 20 dernières années ; le recueil des contraintes globales de Beldiceanu en recense près de 230 [10]. En revanche, très peu de travaux ont été entrepris sur des contraintes globales en domaines continus. Dans le cadre de cette thèse, nous nous somme intéressé aux contraintes globales et stratégies de recherche pour les CSPs continus.

Concevoir une contrainte globale Il existe principalement 2 approches différentes pour mettre en œuvre une contrainte globale :

1. *Inférence de contraintes redondantes* : Ajouter des contraintes supplémentaires au système initial de manière à accélérer ou à renforcer le filtrage par consistance locale. Cette technique ne nécessite pas la définition d'un algorithme dédié.

   Par exemple, considérons 4 variables x_1, x_2, x_3 et x_4 dont les domaines sont respectivement $\boldsymbol{d}_1 = \{1\}$, $\boldsymbol{d}_2 = \{1\}$, $\boldsymbol{d}_3 = \{1, 2, 3, 4\}$ et $\boldsymbol{d}_4 = \{1, 2, 3, 4\}$. On veut que ces variables vérifient la contrainte de cardinalité suivante : chacune des valeurs $\{2, 3, 4\}$ doit être affectée au moins une fois à l'une des variables. Les contraintes sont donc définies par $atleast(\mathcal{X}, 1, 2)$, $atleast(\mathcal{X}, 1, 3)$ et $atleast(\mathcal{X}, 1, 4)$ [1], où $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$.

   Il est clair que ce CSP n'a pas de solutions puisqu'on voudrait que 3 valeurs soient affectées alors que seulement 2 variables peuvent les prendre. Or, un filtrage par arc-consistance ne permet pas de retirer de valeurs dans les domaines de x_3 et x_4, car toutes les valeurs de l'ensemble $\{1, 2, 3\}$ appartiennent à au moins 2 domaines.

   On remarque que si l'on considère ces 3 contraintes simultanément, on peut ajouter une contrainte supplémentaire : $atmost(\mathcal{X}, 1, 1)$. En effet, si 3 valeurs ($\{2, 3, 4\}$) doivent être affectées à 4 variables alors les autres valeurs possibles ($\{1\}$) ne peuvent plus être prises qu'une seule fois ($4 - 3 = 1$). Or l'inconsistance de cette contrainte est facile à vérifier puisque la valeur 1 est prise simultanément par x_1 et x_2.

   Cela montre que l'on peut inférer des contraintes supplémentaires en prenant en compte simultanément un ensemble de contraintes et que ces contraintes additionnelles peuvent améliorer la qualité et l'efficacité du filtrage par arc-consistance classique.

2. *Filtrage dédié* : Utiliser une autre modélisation du problème et définir un algorithme de filtrage des domaines plus fort et plus efficace. La plupart des contraintes globales sont associées à des algorithmes de filtrage puissants. C'est le cas par exemple de la contrainte globale de cardinalité, illustrée précédemment, qui utilise un algorithme basé sur la théorie des flots [105]. L'algorithme de filtrage associé à la contrainte AllDiff [104] modélise le problème par un graphe bi-parti, puis utilise plusieurs algorithmes de décomposition structurelle pour éliminer les valeurs inconsistantes des domaines. En fait, près de 90% des algorithmes de filtrage pour les contraintes globales sur les domaines finis utilisent des algorithmes et des propriétés de la théorie des graphes [11].

Contraintes globales pour les contraintes de distance Nous avons tenté dans cette thèse de concevoir une contrainte globale pour les systèmes de contraintes de distance euclidienne. Ces systèmes sont définis par un ensemble de points et de distances entre certains

[1] L'opérateur $atleast(\mathcal{X}, k, n)$ (resp. $atmost(\mathcal{X}, k, n)$) [118] signifie que la valeur n doit être affectée à au moins (resp. au plus) k variables de $\mathcal{X}$ simultanément

couples de ces points. Le problème est de localiser ces points dans l'espace de manière à ce que les contraintes de distance soient respectées. Les systèmes de contraintes de distance sont présents dans un large champ d'applications (conception optimale de robots, contraintes géométriques, CAO, conformation moléculaire, ...). Comme exemple d'applications on peut citer le manipulateur de type 3-RPR (voir figure 7.7). Ce robot plan est formé de deux triangles : la base ABC et la plate-forme DEF reliés par 3 jambes AD, BE et CF. Le problème de la cinématique directe est de trouver toutes les positions de la plate-forme compatibles avec les longueurs des jambes qui sont les données du problème. Ce problème possède 6 solutions réelles [45].

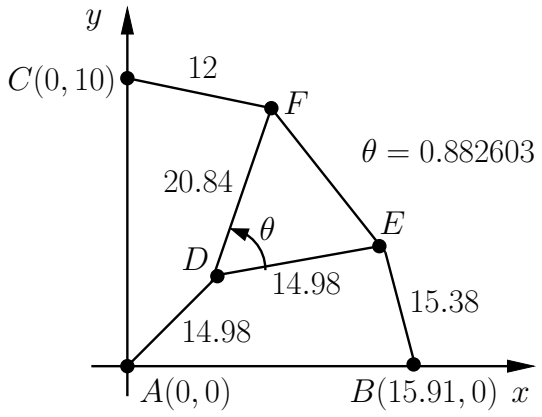


FIG. 1.1 – Un manipulateur planaire de type 3-RPR

Nous avons dans cette thèse exploré les deux voies différentes pour la conception de contraintes globales que l'on a cité précédemment :

1. **Inférence de contraintes redondantes** [2, 3] : Nous avons tenté d'introduire des contraintes additionnelles en nous basant sur des propriétés géométriques élémentaires des systèmes de contraintes de distance. Deux approches sont détaillées dans cette thèse, la première est basée sur l'inférence de contraintes de distance supplémentaires, la deuxième sur l'introduction de points et de distances supplémentaires :
   – *Fermeture du graphe de contraintes* : Étant donné un système de contraintes de distance, certaines distances entre des couples de points sont données soit par une valeur numérique, soit par un intervalle, d'autres distances sont inconnues. L'idée est de trouver un intervalle pour borner toutes les distances manquantes en utilisant les inégalités triangulaires :

   $$\forall i, j, k \in [1..n] : i \neq j \neq k \begin{cases} \delta_{ij} \leq \delta_{ik} + \delta_{jk} \\ \delta_{ij} \geq |\delta_{ik} - \delta_{jk}| \end{cases}$$

   Un algorithme polynômial a été proposé pour résoudre ce problème : l'algorithme BoundSmoothing [31]. Cet algorithme est basé celui de Floyd qui calcule tous les plus courts chemins dans un graphe. On peut d'ailleurs considérer cet algorithme à lui seul comme une contrainte globale sur les systèmes d'inégalités triangulaires.
   La fermeture du graphe de contraintes de distance (voir figure 1.2) ajoute des contraintes redondantes au système initial, ce qui permet dans certains cas de détecter l'inconsistance du système plus rapidement. En effet, si les distances données ne satisfont pas

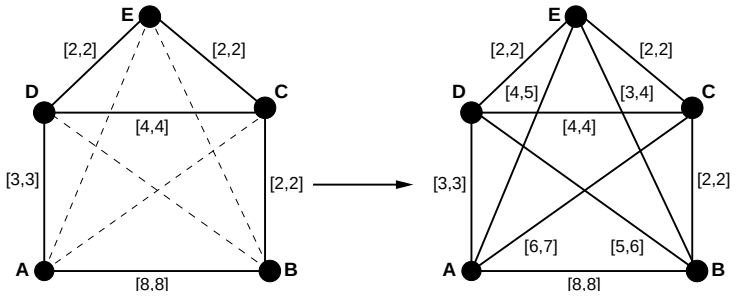


FIG. 1.2 – Fermeture d'un graphe de contraintes de distance

les inégalités triangulaires alors le système est trivialement inconsistant. En général, la fermeture du graphe de distance améliore la propagation des réduction de domaines et donc les performances des techniques de consistance.

– *Introduction de barycentres* : Une fois que le graphe de contraintes a été fermé par la procédure décrite précédemment, les domaines de l'ensemble des distances sont consistants avec le système d'inégalités triangulaires. Cette propriété permet d'introduire une approximation des distances entre le barycentre et chacun des sommets d'un triangle. Ainsi, le système suivant est généré pour chaque triangle $\mathcal{P}_i\mathcal{P}_j\mathcal{P}_k$:

$$\mathcal{C}_{ijk} = \left\{ \begin{array}{rcl} (x_G - x_i)^2 + (y_G - y_i)^2 & = & \frac{1}{9}(3\delta_{ik}^2 + 3\delta_{ij}^2 - \delta_{jk}^2) \\ (x_G - x_j)^2 + (y_G - y_j)^2 & = & \frac{1}{9}(3\delta_{ij}^2 + 3\delta_{jk}^2 - \delta_{ik}^2) \\ (x_G - x_k)^2 + (y_G - y_k)^2 & = & \frac{1}{9}(3\delta_{ik}^2 + 3\delta_{jk}^2 - \delta_{ij}^2) \\ x_G & = & \frac{1}{3}(x_i + x_j + x_k) \\ y_G & = & \frac{1}{3}(y_i + y_j + y_k) \end{array} \right.$$

Ces équations exploitent des propriétés élémentaires sur les triangles et les barycentres. Les distances entre barycentres et sommets encapsulent explicitement les informations données par les distances entre les sommets, mais également et de manière implicite, des informations sur les angles entre les arêtes du triangle. L'ajout de ces contraintes et de ces barycentres au système initial permet un filtrage plus fort des domaines.

2. **Filtrage dédié** [8] : Nous avons spécialisé la contrainte globale Quad [76, 89, 75], dédiée aux systèmes d'équations quadratiques, pour les contraintes de distance. Quad linéarise les termes quadratiques des équations (de la forme xy ou x^2), autrement dit Quad génère un certain nombre d'inéquations linéaires qui approximent l'espace des solutions. Quad utilise ensuite le simplexe sur le système linéaire généré pour réduire les bornes des domaines. Le point clé de cette méthode est que les approximations linéaires produites garantissent la conservation des solutions [89].

   Nous avons introduit une nouvelle technique de linéarisation spécifiquement adaptée aux contraintes de distance. Cette nouvelle technique, nommée QuadDist, ne génère pas des linéarisations pour chaque terme des équations mais globalement pour chaque contrainte de distance (voir figure 1.3). QuadDist définit donc une donc une approximation plus précise que Quad, ce qui augmente la vitesse de convergence de la méthode. D'autre part, contrairement à la Quad, notre linéarisation des contraintes ne nécessite pas l'ajout de variables supplémentaires ce qui réduit la taille des systèmes linéaires résolus par le

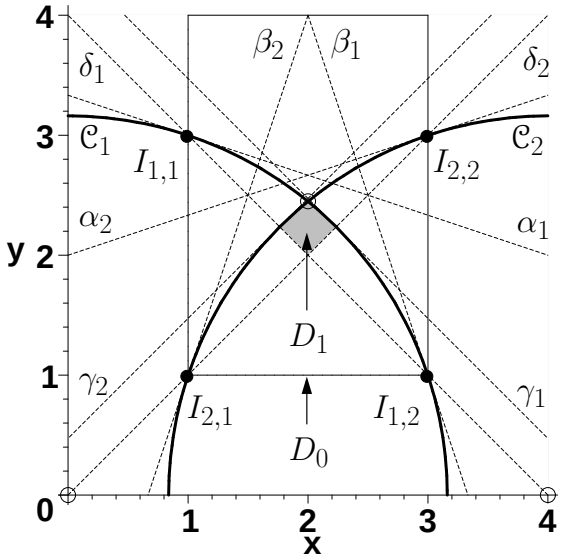


FIG. 1.3 – Exemple de linéarisation par QuadDist.

simplexe. Les résultats expérimentaux montrent que QuadDist résout certains problèmes près de 300 fois plus vite que la Quad.

1.2 Techniques de résolution pour les CSPs continus

La plupart des heuristiques pour améliorer l'algorithme de Branch&Prune n'exploitent pas la sémantique des contraintes pour décomposer les domaines des variables. Par exemple, la stratégie *Round Robin* sélectionne les variables à tour de rôle dans un ordre prédéfini ou encore la stratégie *Largest First* sélectionne la variable dont le domaine est le plus grand. Les stratégies standard de découpe des domaines se basent sur une décomposition de l'espace de recherche en sous-problèmes de même taille. Par exemple, la bisection coupe un domaine en son milieu. Aucune informations sur la sémantique des contraintes n'est utilisée.

Or, l'utilisation de certaines propriétés de équations permet de mieux choisir les variables et de trouver des points de coupe plus pertinents dans les domaines sélectionnés. Par exemple, une des stratégie les plus performantes est de sélectionner la variable qui maximise la *smear function* [66], qui quantifie d'une certaine manière la capacité de filtrage d'une variable. L'idée est de couper la variable qui a le plus de chance de conduire à un filtrage plus fort. Cette fonction est calculée en utilisant des propriétés sur la Jacobienne par intervalles du systèmes et donc est basée sur la sémantique des contraintes.

Dans le même esprit, nous avons défini une heuristique de sélection de domaines et de choix de point de coupe qui tient compte de la sémantique des contraintes. Cette heuristique a été mise en évidence tout d'abord sur les systèmes de contraintes de distance. En effet, nous avons remarqué qu'il était intéressant dans certains cas de décomposer les domaines des variables en utilisant les propriétés de monotonie et de convexité des contraintes de distance. Cette décomposition a fait apparaître des trous dans les domaines des variables, c'est-à-dire des intervalles de valeurs inconsistantes strictement inclus dans les domaines (voir figure 1.4).

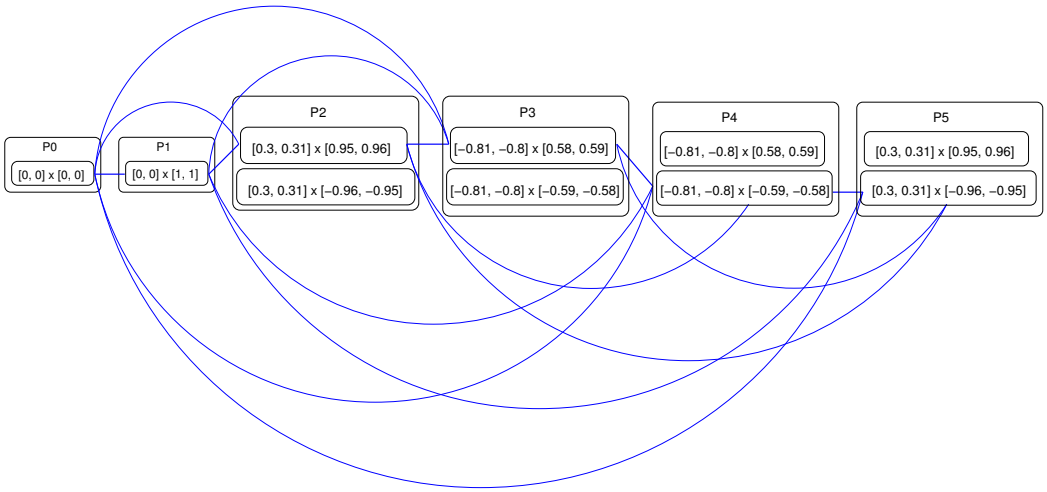


FIG. 1.4 – Décomposition sémantique du problème du pentagone. Les solutions du problème correspondent aux cliques maximales de ce graphe de micro-structure.

DS : Décomposition sémantique pour les contraintes de distance [6, 5, 7] Dans un premier temps, nous avons introduit une technique de Décomposition Sémantique (DS) spécifique aux contraintes de distance et qui identifie et exploite ces trous dans les domaines. Cette technique se place en amont de la résolution proprement dite et décompose le système initial en un certain nombre de sous-problèmes vérifiant des propriétés de convexité et de monotonie. Le principe de DS est de modéliser le CSP continu par un CSP fini dont les valeurs sont des intervalles (voir figure 1.4), puis d'utiliser une technique de résolution comme le maintien d'arc-consistance (MAC [112]) pour engendrer des sous-problèmes. Ces sous-problèmes sont 2B-consistants et vérifient des propriétés de monotonie et de convexité.

MindTheGaps : Une nouvelle stratégie de recherche pour les CSPs continus [9, 4] Dans un deuxième temps, nous avons généralisé cette approche à des contraintes numériques quelconques. En effet, nous avons remarqué que les techniques de consistance locales les plus utilisées (Hull-Consistance [79, 80, 13], Box-consistance [14, 119, 26]) de même que les consistances partielles (kB-consistances [79, 74], Bound-consistance [100]) identifient des trous dans les domaines des variables. Nous avons introduit une nouvelle stratégie de recherche pour les consistances locales, nommé MindTheGaps. Cette technique exploite ces trous à la fois pour choisir la direction de coupe ainsi que pour définir les points de coupe dans le domaine sélectionné. Découper le domaine en éliminant ces trous permet de réduire l'espace de recherche, ce qui n'est pas le cas des techniques de splitting classique comme la bisection. Cela aide le solveurs à éliminer certaines solutions redondantes et ainsi qu'à isoler des solutions disjointes. Des résultats expérimentaux montrent que MindTheGaps améliore de manière significative les performances des algorithmes de résolution classique.

Plan du manuscrit Ce manuscrit est organisé en 3 parties :

I La première partie fait un état de l'art restreint du domaine de la programmation par contraintes. Dans le chapitre 2, les principales techniques de résolution et de filtrage par consistance en domaines finis seront présentées. Le chapitre 3 présente l'arithmétique

des intervalles et l'adaptation des techniques de consistances des domaines finis aux intervalles.

II La deuxième partie présente nos travaux sur les contraintes globales. Le chapitre 4 définit le problème de satisfaction de contraintes de distance, ses applications et les principales approches pour le résoudre. Le chapitre 5 présente la première approche que nous avons explorée, c'est-à-dire l'inférence de contraintes supplémentaires générées à partir des contraintes du système initial. Le chapitre 6 présente QuadDist, une technique de filtrage global spécifique aux contraintes de distance.

III La troisième partie de ce manuscrit présente nos travaux sur les techniques de résolution de CSPs continus. Le chapitre 7 présente DS, une heuristique de recherche dédiée aux contraintes de distance. Enfin, le chapitre 8 présente MindTheGaps, une généralisation de cette heuristique de recherche à des contraintes numériques quelconques.

Première partie

État de l'art

Chapitre 2

CSPs à domaines finis

Les problèmes de satisfaction de contraintes (CSP : Constraint Satisfaction Problem) constituent un cadre simple et formel pour représenter et résoudre des problèmes en intelligence artificielle. De nombreux problèmes en IA, mais également dans d'autres disciplines de l'informatique, peuvent être modélisés par un CSP. La programmation par contraintes (PPC) est un paradigme permettant de formuler une large variété de problèmes, allant du problème d'allocation de ressources et d'ordonnancement à la conception assistée par ordinateur. De nombreuses techniques spécifiques à la PPC on été introduites ; la PPC est aujourd'hui réputée pour sa simplicité et sa capacité à traiter des problèmes très généraux.

Toutefois, savoir si un CSP a une solution ou trouver une ou toutes les solutions d'un CSP reste un problème NP-complet, dans le cas général. Un grand nombre de méthodes ont donc été développées pour tenter de réduire le coût de la résolution d'un CSP. Parmi ces méthodes, les plus populaires sont les techniques de consistance qui tentent de réduire l'espace de recherche avant ou pendant la résolution proprement dite.

Ce chapitre présente de manière succincte les principales approches pour modéliser et explorer l'espace de recherche des CSPs finis. Le but n'est pas de faire un état de l'art complet (pour cela le lecteur pourra se reporter à [83, 88, 63, 71]), mais de donner les idées nécessaire à la compréhension du reste de cette thèse. Les CSPs continus, qui sont au coeur de notre étude, seront décrits dans le chapitre 3.

La section 2.1 définit les CSPs de manière générale, puis la section 2.2 présente les techniques de filtrage par consistance pour réduire l'espace de recherche. La section 2.3 décrit la méthode classique pour résoudre des CSPs à domaines finis : l'algorithme de backtracking. La section 2.4 décrit des algorithmes de résolution qui utilisent un filtrage par consistance pendant la recherche de solutions.

2.1 Définition et exemples

Un problème de satisfaction de contraintes est défini par un ensemble de variables, un ensemble de domaines associés et un ensemble de contraintes sur ces variables. Le domaine d'une variable est un ensemble des valeurs autorisées pour cette variable. Une contrainte spécifie un ensemble d'affectations de variables compatibles. Une solution d'un CSP est une

affectation de toutes les variables dans leurs domaines respectifs qui soit compatible avec toutes les contraintes. Formellement,

Définition 2.1.1 (Constraint Satisfaction Problem [91, 84]) *Un CSP (Constraint Satisfaction Problem) est défini par :*
- $\mathcal{X} = \{x_1, \ldots, x_n\}$, *un ensemble de* n ***variables***.
- $\mathcal{D} = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n\}$, *un ensemble de* ***domaines*** *associés aux variables. Le domaine* $\boldsymbol{d}_i$ *est un ensemble de valeurs autorisées pour la variable* x_i.
- $\mathcal{C} = \{c_1, \ldots, c_m\}$, *un ensemble de* ***contraintes*** *reliant l'ensemble des variables.*

Remarque Ce chapitre présente les outils pour modéliser et résoudre les CSPs finis, dont les domaines sont des ensembles finis de valeurs. Les CSPs continus, dont les domaines sont des intervalles réels, seront présentés dans le chapitre 3. Dans le cas des CSPs finis, les contraintes peuvent être modélisées *en extension*, c'est-à-dire qu'une contrainte $c(x_{i_1}, \ldots, x_{i_k})$ d'arité k est un sous-ensemble du produit cartésien de $\boldsymbol{d}_{i_1} \times \ldots \times \boldsymbol{d}_{i_k}$ contenant les tuples qui satisfont c. Ces contraintes peuvent être également modélisées *en intention* , c'est-à-dire sous la forme d'une équation numérique. On verra dans le chapitre 3 que les contraintes en domaines continus ne sont représentables qu'en intention.

L'exemple suivant montre un CSP à domaines finis avec une représentation des contraintes en intention et en extension :

Exemple 2.1.1 (Exemple de CSP à domaines finis) *Soit le CSP* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *définit par :*
- $\mathcal{X} = \{x, y, z, t\}$,
- $\mathcal{D} = \{D_x : \{-5, 0, 3, 4, 5\},\ D_y : \{-4, -3, -2, 0, 5\},\ D_z : \{2, 3, 4\},\ D_t : \{-5, -4, -3\}\}$,
- $\mathcal{C} = \begin{cases} c_1 : x^2 + y^2 = 25 \\ c_2 : z^2 + t^2 = 25 \\ c_3 : x + z = 7 \\ c_4 : y + t = -7 \end{cases}$ *ou* $\mathcal{C} = \begin{cases} c_1 : (x, y) \in \{(-5, 0), (3, -4), (4, -3), (5, 0)\} \\ c_2 : (z, t) \in \{(3, -4), (4, -3)\} \\ c_3 : (x, z) \in \{(5, 2), (3, 4), (4, 3)\} \\ c_4 : (y, t) \in \{(-3, -4), (-4, -3)\} \end{cases}$

Les solutions de ce CSP sont $(x, y, z, t) = (3, -4, 4, -3)$ *et* $(x, y, z, t) = (4, -3, 3, -4)$.

Notations Dans la suite on utilisera les notations[1] suivantes :

$(\mathcal{X}, \mathcal{D}, \mathcal{C})$	un CSP fini
$\mathcal{X} = \{x_1, \ldots, x_n\}$	L'ensemble des variables
$\mathcal{D} = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_n\}$	L'ensemble des domaines finis associés
$\mathcal{C} = \{c_1, \ldots, c_m\}$	L'ensemble des contraintes
n	le nombre de variables de $\mathcal{X}$ ($n = \lvert\mathcal{X}\rvert$).
$\boldsymbol{d}_i$	le domaine fini associé à la variable x_i.
d	le nombre d'éléments du plus grand domaine de $\mathcal{D}$.
v_i	un élément du domaine fini $\boldsymbol{d}_i$.
m	le nombre de contraintes de $\mathcal{C}$ ($m = \lvert\mathcal{C}\rvert$).
$\mathcal{X}(c)$	l'ensemble des variables de la contrainte c.
$(v_1, \ldots, v_n) \in c$	c est satisfaite lorsque $x_1 = v_1, \ldots, x_n = v_n$.
$(v_1, \ldots, v_n) \notin c$	c n'est est pas satisfaite lorsque $x_1 = v_1, \ldots, x_n = v_n$.

[1] Un récapitulatifs des notations utilisées dans ce manuscrits est présenté à la page x

On utilise souvent une définition plus restreinte des CSPs, celle des CSPs binaires par opposition aux CSPs généraux. Un CSP est dit **binaire** si toutes ses contraintes portent sur au plus 2 variables. Les CSPs **généraux** sont des CSPs contenant au moins une contrainte non binaire.

Les CSPs binaires sont représentés par un graphe de contraintes et de manière plus fine par une micro-structure :

- Le graphe de contraintes associé à un CSP binaire est le graphe dont les sommets correspondent aux variables du CSP et les arêtes relient deux variables liées par une contraintes (Fig. 2.1).
- La micro-structure associée à un CSP binaire est le graphe dont les sommets sont des couples (variable,valeur de son domaine) et dont les arêtes relient les couples compatibles. (Fig. 2.2).

Plus formellement,

Définition 2.1.2 (Graphe associé à un CSP binaire) *Soit $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, un CSP binaire. Le **graphe de contraintes** associé à P est le graphe $G = (\mathcal{S}, \mathcal{A})$ tel que :*

- *L'ensemble $\mathcal{S}$ des sommets de G est défini par $\mathcal{S} = \mathcal{X}$*
- *L'ensemble $\mathcal{A}$ des arêtes de G est défini par :*

$$\mathcal{A} = \{(x_i, x_j) \in \mathcal{S}^2 : \exists c \in \mathcal{C} \text{ t.q. } \mathcal{X}(c) = \{x_i, x_j\}\}$$

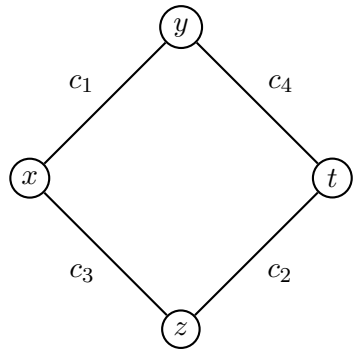


FIG. 2.1 – Graphe de contraintes associé au CSP de l'exemple 2.1.1 page 12

Définition 2.1.3 (Micro-structure d'un CSP binaire) *Soit $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, un CSP binaire. La **micro-structure** associée à P est le graphe $M = (\mathcal{S}, \mathcal{A})$ tel que :*

- *L'ensemble $\mathcal{S}$ des sommets de M est défini par $\mathcal{S} = \{\langle x_i, v_i \rangle : x_i \in \mathcal{X}, v_i \in \boldsymbol{d}_i\}$*
- *L'ensemble $\mathcal{A}$ des arêtes de M est défini par :*

$$\mathcal{A} = \{(\langle x_i, v_i \rangle, \langle x_j, v_j \rangle) \in \mathcal{S}^2 : \exists c \in \mathcal{C} \text{ t.q. } \mathcal{X}(c) = \{x_i, x_j\} \wedge (v_i, v_j) \in c\}$$

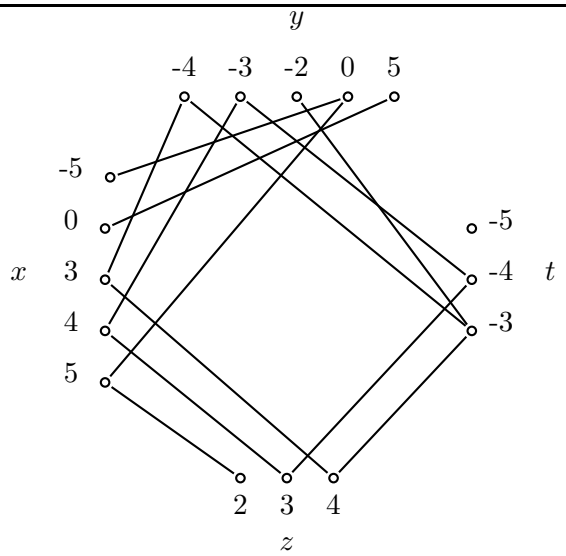


FIG. 2.2 – Micro-structure associée au CSP de l'exemple 2.1.1 page 12. Pour simplifier la figure, seuls les liens entre les couples de variables reliées par une contraintes sont représentées. Les variables x et t d'une part et de y et z d'autre part ne sont liées par aucune contrainte ; les valeurs de x (resp. y) sont donc compatibles avec toutes les valeurs de t (resp. z).

Les CSPs généraux se modélisent par un *hypergraphe de contraintes* qui étend la notion de graphe de contraintes des CSPs binaires. L'*hypergraphe de contraintes* associé à un CSP est l'hypergraphe dont les sommets sont les variables du CSP et les hyperarêtes sont les ensembles des variables liées par une même contrainte. Rossi *et. al.* ont montré qu'il était possible, avec toutefois un coût non négligeable, de convertir un CSP général en CSP binaire [110]. En pratique, il est inconcevable de transformer un système de contraintes non-binaires en un système de contraintes binaires, dû au coût en temps de calcul et en mémoire, en particulier pour les contraintes non-représentables [91].

Définition 2.1.4 (Hypergraphe associé à un CSP général) *Soit $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, un CSP général. L'**hypergraphe de contraintes** associé à P est l'hypergraphe $G = (\mathcal{S}, \mathcal{H})$ tel que :*
- *L'ensemble $\mathcal{S}$ des sommets de G est défini par $\mathcal{S} = \mathcal{X}$*
- *L'ensemble $\mathcal{H}$ des hyperarêtes de G est défini par :*

$$\mathcal{H} = \{(x_{i_1}, \ldots, x_{i_p}) \in \mathcal{S}^p : \exists c \in \mathcal{C} \text{ t.q. } \mathcal{X}(c) = \{x_{i_1}, \ldots, x_{i_p}\}$$

2.2 Filtrage par consistance

Une des approches les plus classiques en intelligence artificielle pour pallier au problème de complexité combinatoire inhérent aux CSPs est d'utiliser des techniques de filtrage par consistance . Ces techniques tentent d'éliminer des valeurs dans les domaines des variables pour lesquels certaines contraintes ne peuvent être satisfaites (*valeurs inconsistantes*). Cela permet de réduire au préalable la taille des domaines et donc de limiter l'effort de recherche.

Un algorithme de filtrage associé à une propriété Φ de consistance consiste à réduire les domaines d'un CSP afin qu'il vérifie Φ. Le CSP résultant de ce filtrage est inclus dans le CSP initial, c'est-à-dire que son espace de recherche a été réduit tout en conservant l'ensemble de ses solutions.

Les filtrages par consistance présentées dans cette section sont à la base des techniques de filtrage de CSPs continus, que nous présenterons dans le chapitre 3. Toutefois, s'il est envisageable pour des domaines finis d'éliminer les valeurs unes à unes, cela n'est pas possible pour des domaines non dénombrable comme les intervalles sur les réels. On verra dans la section 3.5 comment ces techniques de consistance ont été adaptées aux CSPs continus.

2.2.1 Arc-consistance

L'arc-consistance est la plus populaire des consistances. Elle a été définie en 1975 par Waltz pour les CSPs binaires [120], puis étudiée par Mackworth [84] qui a notamment défini la première génération d'algorithme de filtrage par arc-consistance. Cette définition est basée sur la notion de *support*. De manière informelle, on dira qu'une valeur $v_i \in \boldsymbol{d}_i$ possède un support sur une contrainte c s'il existe une instanciation des autres variables dans leurs domaines qui satisfait c lorsque $x_i = v_i$. Le domaine d'une variable x_i est arc-consistant si chacune de ses valeurs possède un support sur toutes les contraintes du CSP. Un CSP est arc-consistant si tous ses domaines sont arc-consistants. Plus formellement,

Définition 2.2.1 (Notion de support) *Soit c une contrainte telle que $\mathcal{X}(c) = \{x_{i_1}, \ldots, x_{i_p}\}$. Soit v_{i_k} une valeur du domaine de la variable $x_{i_k} \in \mathcal{X}(c)$. Le* ***support*** *de v_{i_k} sur c est l'ensemble défini par :*

$$\pi_c^{v_{i_k}} = \{(v_{i_1}, \ldots, v_{i_k}, \ldots, v_{i_p}) \in c / v_{i_1} \in \boldsymbol{d}_{i_1} \ldots v_{i_p} \in \boldsymbol{d}_{i_p}\}$$

Remarque La notation choisie n'est pas standard, mais est très proche de la notion de projection de contraintes que l'on détaillera dans le chapitre 3.

On dira que v_i possède un support sur c si $\pi_c^{v_i} \neq \emptyset$. Au contraire, v_i n'a pas de support sur c si $\pi_c^{v_i} = \emptyset$. Plus généralement, on dira que la valeur v_i d'une variable x_i a un support si $\forall c \in \mathcal{C} / x_i \in \mathcal{X}(c) : \pi_c^{v_i} \neq \emptyset$. La valeur v_i d'une variable x_i n'a pas de support si $\exists c \in \mathcal{C} / x_i \in \mathcal{X}(c) : \pi_c^{v_i} \neq \emptyset$.

Définition 2.2.2 (Arc-consistance [84]) *On dit que le domaine $\boldsymbol{d}_i$ d'une variable x_i est* ***arc-consistant*** *ssi :*

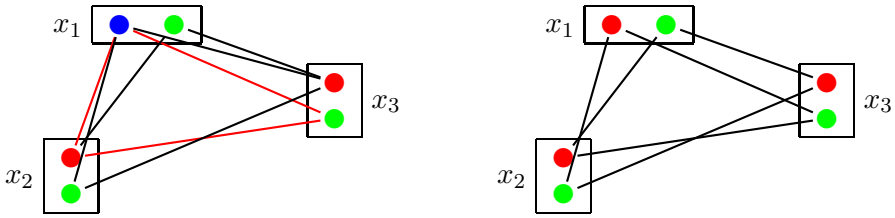
$$\forall v_i \in \boldsymbol{d}_i, \forall c_j \in \mathcal{C} / x_i \in \mathcal{X}(c_j) : \pi_{c_j}^{v_i} \neq \emptyset$$

Un CSP est ***arc-consistant*** *ssi tous ses domaines sont arc-consistants.*

Exemple 2.2.1 *Considérons deux problèmes de coloriage de graphe a trois noeuds, décrits par les CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ et $P' = (\mathcal{X}, \mathcal{D}', \mathcal{C})$ tels que :*

- $\mathcal{X} = \{x_1, x_2, x_3\}$
- $\mathcal{D} = \{\boldsymbol{d}_1 : \{\bullet, \bullet\}, \boldsymbol{d}_2 : \{\bullet, \bullet\}, \boldsymbol{d}_3 : \{\bullet, \bullet\}\}$
- $\mathcal{D}' = \{\boldsymbol{d}_1 : \{\bullet, \bullet\}, \boldsymbol{d}_2 : \{\bullet, \bullet\}, \boldsymbol{d}_3 : \{\bullet, \bullet\}\}$
- $\mathcal{C} = \begin{cases} c_1 : x_1 \neq x_2 \\ c_2 : x_2 \neq x_3 \\ c_3 : x_1 \neq x_3 \end{cases}$

Tous les domaines du CSP P sont arc-consistants (Fig. 2.3(a)). Sur cet exemple, on voit bien que si $x_1 = \bullet$, c_1 (resp. c_3) est vérifiée pour $x_2 = \bullet$ ou pour $x_2 = \bullet$ (resp. $x_3 = \bullet$ ou pour $x_3 = \bullet$). Si $x_1 = \bullet$, c_1 (resp. c_3) est vérifiée pour $x_2 = \bullet$ (resp. $x_3 = \bullet$). La figure 2.3(a) montre également qu'une solution possible de ce CSP est $(\bullet, \bullet, \bullet)$. En revanche, le CSP P' est également arc-consistant (Fig. 2.3(b)), mais n'a pas de solutions.



(a) Un CSP arc-consistant et une de ses solutions(en rouge)

(b) Un CSP arc-consistant mais qui n'a pas de solutions

FIG. 2.3 – Arc-consistance pour un problème de coloriage de graphe

Pour rendre un domaine arc-consistant il suffit d'éliminer les valeurs qui ne satisfont pas la propriété d'arc-consistance. L'élimination de ces valeurs ne modifient pas l'espace des solutions du CSP. Mackworth a défini l'un des premiers algorithmes pour calculer l'arc-consistance d'un CSP binaire : AC-3 [84]. Cet algorithme utilise la procédure Revise (Fig. 2.4) pour éliminer des domaines les valeurs qui n'ont pas de support sur une contrainte.

```
    Revise(in : x_i,c)
1 : m ← false
    %% m indique si le domaine de x_i a été modifié
2 : foreach v_i ∈ d_i do
3 :   if π_c^{v_i} = ∅ then
        %% v_i n'a pas de support sur c
4 :     d_i ← d_i \ {v_i}
5 :     m ← true
6 :   endif
7 : endfor
8 : return(m)
```

FIG. 2.4 – Procédure Revise : élimine du domaine de x_i les valeurs qui n'ont pas de support sur la contrainte c. Revise retourne un booléen (m) qui indique si le domaine a été modifié.

Pour que chaque arc du graphe soit consistant il ne suffit pas d'appeler Revise une seule fois sur tous les domaines. Chaque fois que le domaine $\boldsymbol{d}_i$ d'une variable x_i a été réduit par cette procédure, il faut vérifier si les domaines des variables reliées à x_i par une contrainte sont toujours arc-consistants. En effet, Revise peut éliminer dans $\boldsymbol{d}_i$ l'unique support d'une valeur d'un autre domaine $\boldsymbol{d}_k$; dans ce cas il faut remettre à jour $\boldsymbol{d}_k$. Chaque réduction d'un domaine doit donc être propagée sur l'ensemble du CSP tant qu'une réduction est possible (fermeture par arc-consistance).

Mackworth a proposé un algorithme, nommé AC-3 (Fig. 2.5), pour calculer efficacement le filtrage d'un CSP par arc-consistance [84]. Depuis, diverses variations ont été proposées pour calculer l'arc-consistance d'un CSP, comme AC-4 [90], AC-5 [117], AC-6 [16], AC2001 [121] ... Plus récemment, J-C. Régin [107] a proposé une nouvelle version AC-*, qui factorise en

quelque sorte les comportements de ces différents algorithmes. AC-* est générique, adaptatif et configurable et permet de combiner de manière originale les principaux atouts des précédentes versions. Nous avons choisi de présenter AC-3 car il est à la base des techniques de consistances locales en domaines continus, que nous détaillerons dans le chapitre 3.

AC-3 utilise la procédure Revise décrite dans la figure 2.4. Chaque fois que le domaine d'une variable x_i est modifié par la procédure Revise (ligne 6), AC-3 doit mettre à jour les domaines de toutes variables x_k reliées à x_i par une contrainte autre que celle qu'il vient de considérer (ligne 7).

	AC-3(in-out :$P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$)
1 :	$Q \leftarrow \{\langle x_i, c_j \rangle : x_i \in \mathcal{X}(c_j)\}$
2 :	**while** $Q \neq \emptyset$ **do**
3 :	extraire $\langle x_i, c \rangle$ de Q
6 :	**if** Revise(x_i,c_j) **then**
7 :	$Q \leftarrow Q \cup \{\langle x_k, c \rangle : c \neq c_j \wedge x_i, x_k \in \mathcal{X}(c)\}$
8 :	**endif**
9 :	**endwhile**
10 :	**return**

FIG. 2.5 – Algorithme de propagation AC-3 [84]

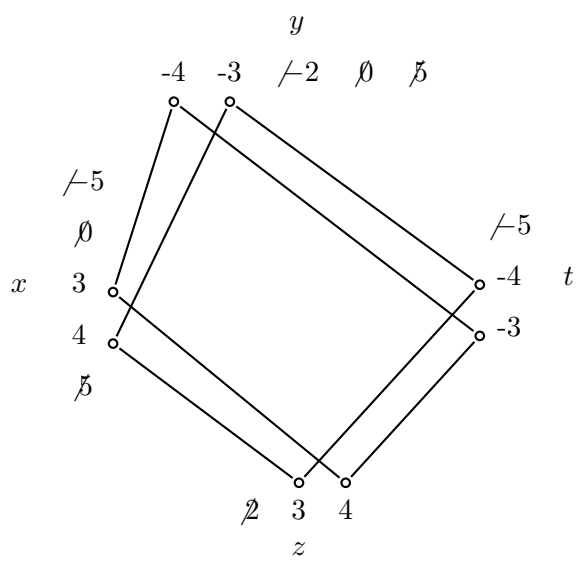


FIG. 2.6 – Filtrage par arc-consistance sur le CSP de l'exemple 2.1.1 page 12

La section suivante présente les principales techniques de consistances partielles.

2.2.2 Consistances partielles

La consistance d'arc permet de réduire l'espace de recherche d'un CSP mais ne suffit pas à résoudre le CSP. Afin de réduire davantage l'espace de recherche, des degrés de consistances plus fortes ont été introduite à la fin des années 70 par Freuder [41] : les k-consistances (Déf. 2.2.3) et les k-consistances fortes (Déf. 2.2.4) :

Définition 2.2.3 (k-consistance [41]) *Un CSP est* ***k-consistant*** *ssi toute instanciation partielle consistante de $k-1$ variables peut être étendue à une instanciation partielle consistante de k variables.*

Définition 2.2.4 (k-consistance forte [41]) *Un CSP est* ***fortement k-consistant*** *ssi il est j-consistant pour tout $j \leq k$.*

Notons que la consistance de noeud est équivalente à la 1-consistance, l'arc-consistance est équivalente à la 2-consistance et la consistance de chemin est équivalente à la 3-consistance.

Des algorithmes pour calculer la k-consistance d'un CSP ont été définis [29, 42]. Si un CSP à n variables est fortement n-consistant alors il n'est plus nécessaire de recourir à un algorithme de backtrack pour le résoudre. Cependant la complexité en temps de calcul au pire des cas pour calculer la n-consistance forte est rédhibitoire. En fait, les k-consistance pour $k > 3$ sont très peu utilisées dû à la complexité exponentielle des algorithmes qui les calculent. De plus, la k-consistance n'est, en général, pas suffisante pour résoudre le CSP ; il faut donc encore faire appel au backtracking. Dans certains cas, selon la structure du graphe de contraintes, la k-consistance peut suffire à résoudre le CSP. D'autres techniques ont été élaborées pour réduire l'espace en recherche en utilisant les propriétés du graphe de contraintes (voir [71] pour plus de détails).

La section suivante présente les techniques classiques de résolution d'un CSP à domaines finis.

2.3 Résolution classique de CSPs finis

Résoudre un CSP consiste à trouver un ensemble de valeurs à affecter aux variables permettant de satisfaire l'ensemble des contraintes. La définition générale d'un CSP inclut par exemple le problème de 3-satisfaisabilité, par conséquent la résolution de CSPs est en général un problème NP-complet.

Plusieurs approches ont été développées pour résoudre des CSPs finis. Parmi ces méthodes on distingue les méthodes *complètes*, qui recherchent toutes les solutions, et les méthodes *incomplètes* comme la *recherche locale* qui se limitent à la recherche d'une seule solution. Nous intéresserons dans ce chapitre à la première catégorie.

Plus formellement, une instanciation partielle est un ensemble d'affectation de variables à une valeur de leur domaine :

Définition 2.3.1 (Instanciation partielle [34]) *Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP fini. Une* ***instanciation partielle*** *$I_{\mathcal{Y}}$ sur le sous-ensemble de variables $\mathcal{Y} \subseteq \mathcal{X}$ est définie par :*

$$I_{\mathcal{Y}} = \{\langle x_i, v_i \rangle : x_i \in \mathcal{Y} \wedge v_i \in \boldsymbol{d}_i\}$$

Une instanciation partielle sur l'ensemble de variables $\mathcal{Y}$ est *localement consistante* si elle satisfait chacune des contraintes mettant en jeu un ensemble de variables inclus dans $\mathcal{Y}$:

Définition 2.3.2 (Consistance locale [34]) *Soient $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP fini, $\mathcal{Y}$ un sous-ensemble de $\mathcal{X}$. Une instanciation partielle $I_{\mathcal{Y}}$ sur l'ensemble de variables $\mathcal{Y}$ est* ***localement consistante*** *ssi :*

$$\forall c \in \mathcal{C} \text{ t.q. } \mathcal{X}(c) = \{x_{i_1}, \ldots, x_{i_p}\} \subseteq \mathcal{Y} : (v_{i_1}, \ldots, v_{i_p}) \in c,$$

où $\langle x_{i_k}, v_{i_k} \rangle \in I_{\mathcal{Y}}, \forall k \leq p$.

Remarque De manière similaire, une instanciation partielle sur l'ensemble de variables $\mathcal{Y}$ est *inconsistante* si il existe une contrainte c telle que $\mathcal{X}(c) \subseteq \mathcal{Y}$ qui ne soit pas satisfaite.

Une solution d'un CSP est une affectation de toutes ses variables par des valeurs de leur domaine qui satisfait toutes les contraintes :

Définition 2.3.3 (Solution[34]) *Une* ***solution*** *d'un CSP* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *est une instanciation localement consistante sur* $\mathcal{X}$.

L'algorithme le plus naïf pour trouver toutes les solutions d'un CSP consiste à générer toutes les combinaisons possibles de valeurs. Les combinaisons sont alors testées ; celles qui satisfont toutes les contraintes sont des solutions du CSP. Cet algorithme, également appelé *Generate & Test*, explore un espace de recherche dont la taille est celle du produit cartésien des domaines.

Une méthode plus efficace combine une recherche arborescente en profondeur d'abord avec des retours-arrières : l'algorithme de *backtracking*. Une première citation de cet algorithme est tirée d'une œvre d'un mathématicien français, Edouard Lucas en 1891 [82]. Cet algorithme a été redécouvert à plusieurs reprises notamment par [44].

L'algorithme de *backtracking* construit une instanciation partielle des variables, en sélectionnant séquentiellement les variables et en leur attribuant une valeur dans leur domaine. Chaque fois qu'une nouvelle valeur est affectée à une variable, un test de consistance est effectué :

- Si au moins une contrainte est violée, la variable courante est affectée à la prochaine valeur de son domaine. S'il ne reste plus de valeurs dans le domaine de la variable courante, l'algorithme effectue un *retour-arrière* , ou *backtrack*, qui consiste à revenir à la variable précédemment instanciée.
- Si toutes les contraintes sont satisfaites, l'instanciation se poursuit avec la variable suivante. Une solution est obtenue lorsque toutes les variables ont été instanciées.

L'ordre dans lequel les variables et les valeurs sont choisies peut influencer de manière significative sur les performances de la recherche. Par exemple, pour trouver une première solution du CSP de l'exemple 2.1.1, l'algorithme de backtrack, avec un ordre alphabétique sur les variables et un ordre croissant sur les valeurs, nécessite 4 retours-arrières et 26 tests de consistance dont 18 échecs (Fig. 2.7). Différentes heuristiques de choix de variables et de choix de valeurs seront présentées dans la section 2.4.

Bien que le *backtracking* soit toujours plus efficace que le *Generate & Test*, il n'en reste pas moins très coûteux et s'adapte mal aux problèmes de grande taille. En effet, l'espace de recherche est parcouru aveuglément sans tenir compte d'informations qui permettrait de réduire le temps de calcul. Ce manque d'efficacité provient essentiellement du fait que le test de consistance peut échouer pour la même raison dans deux espaces de recherche différents.

Supposons par exemple qu'une contrainte unaire sur une variable v interdise une certaine valeur a. Chaque fois que v sera instanciée à a, le test de consistance échouera à cause de cette contrainte unaire, mais cette information ne sera pas prise en compte dans le reste de la recherche. Ce phénomène est qualifié d'*inconsistance de noeud* dans [84] et peut être résolu en éliminant des domaines, préalablement à la recherche, les valeurs incompatibles avec les contraintes unaires.

Avec une contrainte binaire entre deux variables u et v, le même phénomène se produit lorsque u est instanciée à une valeur a et qu'il n'existe aucune valeur compatible dans le domaine de v. Chaque fois que u sera instancié à a, le test de consistance échouera pour la même raison après avoir énuméré le domaine de v dans sa totalité : c'est l'*inconsistance d'arc*

$\langle x, -5\rangle$ ✓	$\langle y, -4\rangle$ •		
	$\langle y, -3\rangle$ •		
	$\langle y, -2\rangle$ •		
	$\langle y, 0\rangle$ ✓	$\langle z, 2\rangle$ •	
		$\langle z, 3\rangle$ •	
		$\langle z, 4\rangle$ •	
	$\langle y, 5\rangle$ •		
$\langle x, 0\rangle$ ✓	$\langle y, -4\rangle$ •		
	$\langle y, -3\rangle$ •		
	$\langle y, -2\rangle$ •		
	$\langle y, 0\rangle$ •		
	$\langle y, 5\rangle$ ✓	$\langle z, 2\rangle$ •	
		$\langle z, 3\rangle$ •	
		$\langle z, 4\rangle$ •	
$\langle x, 3\rangle$ ✓	$\langle y, -4\rangle$ ✓	$\langle z, 2\rangle$ •	
		$\langle z, 3\rangle$ •	
		$\langle z, 4\rangle$ ✓	$\langle t, -5\rangle$ •
			$\langle t, -4\rangle$ •
			$\langle t, -3\rangle$ ✓

FIG. 2.7 – Algorithme de backtrack sur le CSP de l'exemple 2.1.1 page 12

[84]. Ce problème peut également être résolu en calculant une fermeture par *arc-consistance* du CSP.

Ce raisonnement s'étend à 3, 4, ..., k variables, ce qui a engendré des familles d'algorithmes de consistances partielles, comme les k-consistances [41].

Les techniques de consistances permettent de réduire de manière effective les domaines des variables et donc de réduire l'espace de recherche. En règle général elles ne dispensent pas de l'utilisation d'un algorithme de recherche comme le backtrack. La section suivante présente différentes façon de combiner consistances locales et backtracking, ainsi que la plupart des heuristiques de recherche utilisées.

2.4 Amélioration de l'algorithme de recherche

Nous avons présenté deux techniques complémentaires pour résoudre des CSPs finis : le backtracking et le filtrage par consistance. Dans la première, différentes instanciations de variables sont testées jusqu'à trouver une solution. Dans la deuxième, les valeurs inconsistantes (à différents degrés) sont éliminées des domaines. On peut en effet combiner avantageusement les filtrages par consistance et l'algorithme de recherche. MAC [112], la technique la plus efficace pour résoudre les CSPs finis de manière générale, consiste à maintenir l'arc-consistance du CSP durant la recherche de solutions. Parmi les autres combinaisons qui ont été proposées, deux approches se distinguent :

– Les approches prospectives (*Lookahead*) : Après chaque instanciation d'une variable, le *Lookahead* consiste à regarder dans les domaines des variables non encore instanciées s'il est possible d'éliminer des valeurs incompatibles avec l'instanciation courante. L'in-

compatibilité de ces valeurs peut être vérifiée à différents niveaux, ce qui a engendré différents algorithmes présentés dans [50] puis classifiés dans [94]. Les plus connus sont le *forward-checking* (FC), le *partial lookahead* (PL), le *full lookahead* (FL) et le *really full lookahead* (RFL). FC a longtemps été considéré comme la technique la plus efficace pour résoudre des CSPs finis. Plus récemment, Sabin *et. al.* ont développé une technique qui maintient la consistance d'arc après chaque nouvelle instanciation [112]. Il a été observé expérimentalement que cette technique, nommé MAC (Maintaining Arc-Consistency), est la méthode la plus efficace pour résoudre les CSPs finis difficiles [112, 17].

– Les approches rétrospectives (*Lookback*) : Par exemple, le *backchecking*[50] ou le *backmarking* [43] tirent parti des informations contenues lors des échecs de consistance afin de sélectionner un meilleur point de retour. Il y a également d'autres techniques comme le *conflict-set* [32] ou le *backjumping*[33] : Lors d'un backtrack, plutôt que de revenir à la dernière variable instanciée, on revient à la variable qui a causé l'échec du test de consistance.

Dans l'algorithme de backtracking, l'ordre dans lequel les variables sont instanciées peut influencer de manière significative les performances de la résolution. L'ordre dans lequel les valeurs dans les domaines sont affectées peut également jouer un rôle déterminant dans la résolution. Cela a donné lieu à deux types d'heuristiques :

– Ordre de sélection des variables : L'une des méthodes les plus connues est de réorganiser dynamiquement les variables de manière à instancier en premier celle dont le domaine contient le moins de valeurs [18, 50].
– Ordre de sélection des valeurs : On peut par exemple choisir en premier les valeurs les moins contraintes en premier [50, 101, 35].

Les performances d'un algorithme de recherche de solutions d'un CSP fini dépendent donc essentiellement de deux facteurs : les filtrages, qui réduisent l'espace de recherche, et les heuristiques, qui permettent de parcourir cet espace plus intelligemment. Les filtrages forts réduisent grandement la taille de l'espace de recherche mais souvent au détriment des performances. On utilise donc en général des filtrages locaux comme l'arc-consistance. Cependant, le manque d'expressivité des contraintes et la faiblesse de ces filtrages locaux réduisent l'efficacité de la recherche de solutions. Il est donc utile de trouver des algorithmes de filtrage à la fois puissants et efficaces. C'est ce qui est à l'origine des contraintes globales qui sont présentées dans la section suivante.

2.5 Contraintes globales

Les filtrages locaux, comme le filtrage par arc-consistance, sont bien souvent trop faibles car ce sont des algorithmes généraux qui ne tiennent pas compte de la spécificité des contraintes. D'autre part, l'arc-consistance considère les contraintes séparément (localité) alors que certaines contraintes n'ont de sens que lorsqu'elles s'expriment toutes ensembles. Il est donc nécessaire dans ce cas de prendre en compte ces contraintes simultanément.

Une contrainte globale est une contrainte qui exprime un ensemble de contraintes élémentaires. Plus précisément, une contrainte globale est la conjonction d'un ensemble de relations :

Définition 2.5.1 (Contrainte globale) *Une contrainte globale C est définie par la conjonction d'un ensemble de contraintes $\{c_1, \ldots, c_n\}$:*

$$C = c_1 \wedge \ldots \wedge c_n$$

Par opposition aux techniques de consistances locales, les contraintes globales utilisent la simultanéité des contraintes et peuvent donc effectuer de meilleurs filtrages. D'autre part, les contraintes globales expriment de manière plus compacte un ensemble de contraintes.

L'une des contraintes globales les plus anciennes est la contrainte AllDiff [104]. Cette contrainte exprime le fait que toutes les variables du CSP doivent être différentes deux à deux. Autrement dit, pour un ensemble $\mathcal{X}$ de n variables, AllDiff est la conjonction des $n(n-1)$ contraintes de la forme $x_i \neq x_j$, avec $x_i, x_j \in \mathcal{X}$ et $i \neq j$.

Avec la contrainte AllDiff, il n'est plus nécessaire de représenter explicitement les contraintes, c'est à dire, pour chaque contrainte l'ensemble des couples qui la satisfont, d'où un gain en mémoire non négligeable. La force de la contrainte AllDiff est l'algorithme de filtrage par consistance d'arc qui lui est associé. Cet algorithme permet d'obtenir un filtrage des domaines plus fort qu'une réduction par arc-consistance sur le CSP binaire équivalent.

Par exemple, supposons que l'on ait un CSP à 3 variables x, y et z ayant pour domaine $\boldsymbol{d}_x = \boldsymbol{d}_y = \boldsymbol{d}_z = a, b$, avec l'ensemble de contraintes binaires $\{x \neq y, x \neq z, y \neq z\}$. Sous cette forme, l'arc-consistance ne permet pas de réduire les domaines (voir exemple 2.2.1 page 15). Pourtant ce système n'a pas de solution. L'algorithme de filtrage associé à la contrainte AllDiff permet de détecter cette inconsistance.

L'algorithme de filtrage AllDiff[2] utilise une modélisation du problème par un graphe biparti, puis utilise plusieurs techniques de recherche opérationnelle pour éliminer les valeurs inconsistantes des domaines. Notamment, il utilise des techniques issues de la théorie des graphes comme le couplage maximum, la décomposition de Dulmage-Mendelson et les composantes fortement connexes. Il a été montré que cet algorithme calcule une réduction optimale des domaines [98]. La figure 2.8 montre une illustration de l'exécution de cet algorithme.

D'autres contraintes globales ont été introduites, comme entre autres, *Global Cardinality* [105], *Symetric AllDiff* [106], *Cumulative* [12]. Beldiceanu en recense près de 230 dans son recueil des contraintes globales [10]. Le point commun des ces différentes méthodes, pour près de 90% d'entre elles, est le fait qu'elle utilisent des propriétés structurelles des graphes [11]

[2]Par abus de langage, une contrainte globale désigne souvent à la fois l'ensemble des contraintes et l'algorithme de filtrage correspondant.

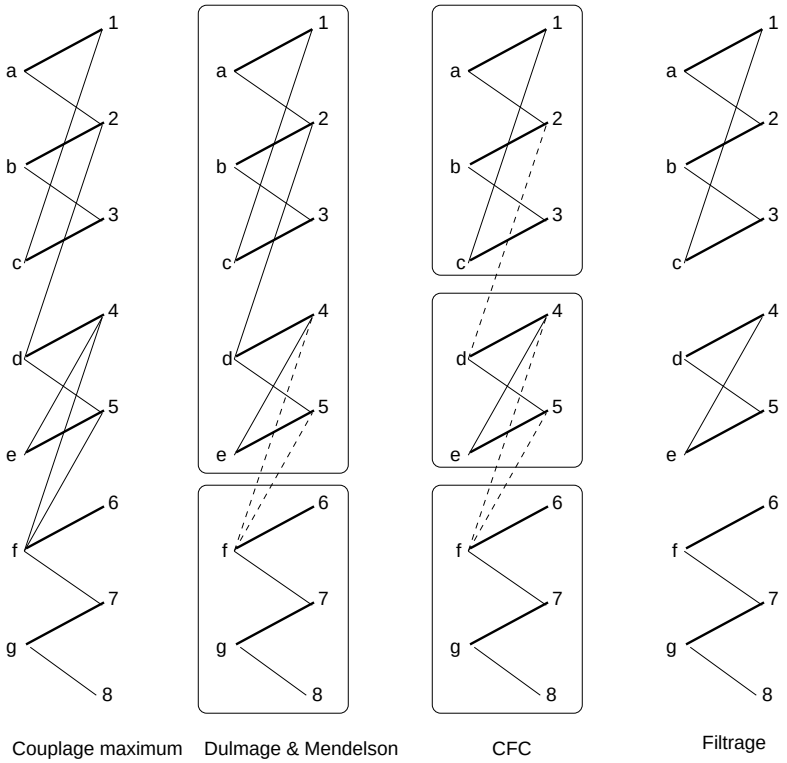


FIG. 2.8 – Illustration d'un filtrage par la contraint globale AllDiff

Chapitre 3

CSPs à domaines continus

La problématique des CSPs continus est radicalement différente de celle des CSP finis présentée dans le chapitre précédent. Un CSP continu est un CSP dont les domaines sont des intervalles, c'est-à-dire des ensembles non dénombrables de valeurs réelles contiguës. Les techniques de résolution de CSP finis basées sur l'énumération de toutes les valeurs des domaines combinée avec l'algorithme de backtracking sont donc inutilisables en pratique. D'autre part, les nombres réels ne sont pas représentables en machine. L'arithmétique des ordinateurs n'autorise qu'une approximation des nombres réels : les nombres flottants.

Une autre manière de représenter les domaines continus est d'utiliser une approximation des intervalles sur les réels par des intervalles dont les bornes sont des flottants. L'arithmétique des intervalles fournit des bases solides pour l'édification de techniques de consistance pour les CSPs continus.

Ce chapitre présente les CSPs à domaines continus et les différentes approches pour les résoudre. La section 3.1 définit les CSPs continus et introduit la notation utilisée dans le reste de ce manuscrit. La section 3.3 donne quelques notions d'arithmétique des intervalles qui seront utiles pour la compréhension de la suite. La section 3.2 décrit brièvement les principales méthodes issues des mathématiques conventionnelles pour la résolution de CSPs continus. La section 3.4 détaille le schéma général de l'algorithme de recherche de solutions classique, l'algorithme Branch&Prune. Les sections 3.5 et 3.6 présentent les techniques de consistance et les contraintes globales pour les CSPs continus. Enfin, d'autres techniques de résolution seront présentées brièvement dans la section 3.7.

3.1 Définition et exemple

Un CSP continu est défini de la manière suivante :

Définition 3.1.1 (CSP continu) *Un **CSP continu**, ou CSP à domaines continus, est défini par :*

- $\mathcal{X} = \{x_1, \ldots, x_n\}$, *un ensemble de* n ***variables***.
- $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, *un ensemble de* ***domaines***. *Le domaine* $\boldsymbol{x}_i$ *est un intervalle contenant l'ensemble des valeurs autorisées pour la variable* x_i
- $\mathcal{C} = \{c_1, \ldots, c_m\}$, *un ensemble de* ***contraintes*** *reliant l'ensemble des variables* $\mathcal{X}$.

Remarque Les contraintes sont modélisées (*en intention*) par des équations numériques de la forme :

$$c_j : f_j(x_1, \ldots, x_n) \bullet 0, \text{avec } \bullet \in \{<, \leq, =, \geq, >\}$$

Les domaines des variables étant non dénombrables, la modélisation des contraintes *en extension* n'est pas représentable en machine. Toutefois, une contrainte, au sens relationnel du terme, désigne l'ensemble des tuples qui la satisfont : x

$$c_j : \{(v_1, \ldots, v_n) : f_j(v_1, \ldots, v_n) \bullet 0\}, \text{avec } \bullet \in \{<, \leq, =, \geq, >\}$$

On conservera donc la notation des CSPs finis : $(v_1, \ldots, v_n) \in c$ si c est satisfaite lorsque $x_1 = v_1, \ldots, x_n = v_n$. Les solutions d'un CSP sont définies par l'intersection de toutes les contraintes :

$$S = \bigcap_{i=1}^{i=m} c_i \cap (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n),$$

où $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ désigne le vecteur intervalle - ou bôite - constitué par le produit cartésien des domaines.

Exemple 3.1.1 (Exemple de CSP à domaines continus) *Soit le CSP* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *défini par :*

- $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$,
- $\mathcal{D} = \{\boldsymbol{x}_1 = [-5, 5],\ \boldsymbol{x}_2 = [-4, 5],\ \boldsymbol{x}_3 = [2, 4],\ \boldsymbol{x}_4 = [-5, -3]\}$,
- $\mathcal{C} = \begin{cases} c_1 : x_1^2 + x_2^2 = 25 \\ c_2 : x_3^2 + x_4^2 = 25 \\ c_3 : x_1 + x_3 = 7 \\ c_4 : x_2 + x_4 = -7 \end{cases}$

Un CSP continu peut être résolu en utilisant une grande variété de méthodes de résolution dont des techniques issues de l'analyse numérique, de la programmation linéaire et non-linéaire, des techniques algébriques, des algorithmes évolutionnaires, de l'analyse par intervalles et des techniques de consistance. Ce chapitre est axé essentiellement sur les deux dernières catégories de méthodes. Dans la section 3.4, d'autres méthodes de résolution seront présentées.

3.2 Méthodes algébriques et numériques

Il existe différentes approches issues des mathématiques conventionnelles pour résoudre des CSPs continus : des méthodes algébriques ou numériques.

Les systèmes d'équations linéaires à coefficients réels sont relativement bien traités par ces outils mathématiques, notamment par la programmation linéaire ou l'algèbre linéaire. Par exemple, la méthode du simplexe [47] permet d'optimiser un critère linéaire sur des variables soumises à un système d'inégalités linéaires. La méthode d'élimination de Gauss est une méthode qui permet de trouver systématiquement les solutions d'un système linéaire bien-contraint. Ces outils sont disponibles dans un large panel de librairies mathématiques. Ces méthodes peuvent être utilisées avec des nombres rationnels ou des nombres flottants ; dans ce dernier cas, les résultats peuvent être entâchés d'erreurs du fait des problèmes d'arrondi.

En revanche, les systèmes non-linéaires offrent plus de résistance aux techniques de programmation mathématique :

– Les méthodes algébriques se limitent au contraintes polynômiales et ne sont pas adaptées pour traiter des systèmes de grande taille. C'est le cas par exemple des solveurs basés sur les bases de Gröbner [23, 24], qui permettent de calculer une formule exacte pour les solutions d'un système d'équations polynômiales. Toutefois, une large classe de problèmes peuvent être résolus par des bases de Gröbner. A l'heure actuelle l'algorithme le plus efficace est F_4 [40], disponible sur la plate-forme Magma.
– Les méthodes numériques permettent de calculer toutes les "solutions" pour tous les systèmes, mais elles ne convergent pas de manière systématique et peuvent avoir un comportement chaotique, notamment en présence de singularités.
  Plusieurs méthodes numériques, comme par exemple la méthode de Newton, ont été étendues aux intervalles afin de pallier aux erreurs d'approximations liés à l'arithmétique sur les flottants. L'arithmétique des intervalles permet de calculer une enveloppe externe des solutions et donc de garantir la correction des résultats.

La programmation par contraintes en domaines continus et l'analyse par intervalles apportent différentes contributions pour la résolution de systèmes de contraintes. Les CSPs sont plus informés qu'un système général d'équations ou d'inéquations car l'espace de recherche des solutions est borné. Ces informations peuvent dans certains cas faciliter la résolution. Les techniques de consistances en domaines continus sont des méthodes générales, c'est-à-dire qu'elle peuvent traiter tout type de contraintes numériques, ce qui n'est pas le cas des méthodes algébriques par exemple. Enfin, l'arithmétique des intervalles permet de traiter des systèmes avec incertitudes sur les paramètres, ce qui n'est pas le cas des méthodes conventionnelles.

La section suivante présente les bases de l'arithmétique des intervalles.

3.3 Arithmétique des intervalles

L'objectif de cette section n'est pas de faire une description complète de l'arithmétique des intervalles et de ses propriétés, mais d'énoncer les définitions nécessaires à la compréhension du reste de ce manuscrit. Le lecteur pourra se reporter à [92, 1, 95, 49, 67, 58] pour plus détails à ce sujet.

3.3.1 Définitions et notations

Soit $\overline{\mathbb{R}}$ l'ensemble des nombre réels $\mathbb{R}$ étendu aux valeurs infinies $\{-\infty, +\infty\}$ et soit $\overline{\mathbb{F}} \subset \overline{\mathbb{R}}$ le sous-ensemble des réels correspondants aux nombres flottants dans un format donné. L'intervalle *fermé* $\boldsymbol{x} = [\underline{x}, \overline{x}]$ dénote l'ensemble des valeurs réelles x telles que $\underline{x} \leq x \leq \overline{x}$. Les intervalles *ouverts* seront dénotés par $]\underline{x}, \overline{x}[= \{x \in \mathbb{R} : \underline{x} < x < \overline{x}\}$. On dira que $\boldsymbol{x}$ est un intervalle *dégénéré* si $\underline{x} = \overline{x}$, et on notera x pour $[x, x]$.

On notera $\mathbb{IR}$ l'ensemble des intervalles à bornes dans $\mathbb{R}$. Les nombres réels n'étant pas tous représentables en machine, les intervalles réels sont approximés par des intervalles dont les bornes sont des nombres flottants. L'ensemble des intervalles à bornes dans $\mathbb{F}$ sera noté $\mathbb{IF}$. Dans la suite, on considérera que les intervalles sont des intervalles de $\mathbb{IF}$. Les variables seront notées x, y et les vecteurs de variables par X, Y. Les intervalles seront notés $\boldsymbol{x}$, $\boldsymbol{y}$ et les vecteurs d'intervalles - ou *boîtes* - par $\boldsymbol{X}$, $\boldsymbol{Y}$, éventuellement indicées.

Notations Dans la suite on utilisera les notations[1] suivantes :

$\mathbb{R}$	L'ensemble des nombre réels
$\overline{\mathbb{R}}$	$\mathbb{R}$ étendu à $\{-\infty, +\infty\}$
$\mathbb{F}$	L'ensemble des nombre flottants dans un format donné ($\overline{\mathbb{F}} \subset \overline{\mathbb{R}}$).
$\overline{\mathbb{F}}$	$\mathbb{F}$ étendu à $\{-\infty, +\infty\}$
$[\underline{x}, \overline{x}]$	un intervalle fermé (l'ensemble des valeurs réelles x telles que $\underline{x} \leq x \leq \overline{x}$)
$]\underline{x}, \overline{x}[$	un intervalle ouvert (l'ensemble des valeurs réelles x telles que $\underline{x} < x < \overline{x}$)
$m(\boldsymbol{x})$	le milieu de l'intervalle $\boldsymbol{x}$ ($m(\boldsymbol{x}) = (\overline{x} + \underline{x})/2$)
$w(\boldsymbol{x})$	la taille de l'intervalle $\boldsymbol{x}$ ($w(\boldsymbol{x}) = \overline{x} - \underline{x}$)
$\mathbb{IR}$	l'ensemble des intervalles à bornes dans $\overline{\mathbb{R}}$.
$\mathbb{IF}$	l'ensemble des intervalles à bornes dans $\overline{\mathbb{F}}$.
x, y, z	des variables réelles
$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$	des intervalles
X, Y, Z	des vecteurs de variables réelles
$\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$	des vecteurs d'intervalles - ou *boîtes*
$\boldsymbol{X} = \emptyset$	si une des composantes $\boldsymbol{x}_i$ de $\boldsymbol{X}$ est vide
$w(\boldsymbol{X})$	La taille du plus grand intervalle qui compose $\boldsymbol{X}$

3.3.2 Opérateurs arithmétiques

L'arithmétique des intervalles est une *extension* de l'arithmétique sur les réels [92]. L'extension aux intervalles $\odot$ d'un opérateur sur les réels $\cdot$ doit respecter la propriété suivante :

$$\boldsymbol{x} \odot \boldsymbol{y} = \{x \cdot y : x \in \boldsymbol{x}, y \in \boldsymbol{y}\}$$

Les extensions aux intervalles des opérateurs arithmétiques de base sur les réels ont été définis de la manière suivante :

Définition 3.3.1 (Opérateurs arithmétiques de base sur les intervalles [92]) *Soient $\boldsymbol{x}$ et $\boldsymbol{y}$ deux intervalles, les opérateurs arithmétiques de base sur les intervalles sont définis de la manière suivante :*

$$\begin{array}{lcll}
[\underline{x}, \overline{x}] \oplus [\underline{y}, \overline{y}] & = & [\underline{x} + \underline{y}, \overline{x} + \overline{y}] & \\
[\underline{x}, \overline{x}] \ominus [\underline{y}, \overline{y}] & = & [\underline{x} - \overline{y}, \overline{x} - \underline{y}] & \\
[\underline{x}, \overline{x}] \otimes [\underline{y}, \overline{y}] & = & [min(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}), max(\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y})] & \\
[\underline{x}, \overline{x}] \oslash [\underline{y}, \overline{y}] & & [min(\frac{\underline{x}}{\overline{y}}, \frac{\underline{x}}{\underline{y}}, \frac{\overline{x}}{\overline{y}}, \frac{\overline{x}}{\underline{y}}), max(\frac{\underline{x}}{\overline{y}}, \frac{\underline{x}}{\underline{y}}, \frac{\overline{x}}{\overline{y}}, \frac{\overline{x}}{\underline{y}})] & \textit{si } 0 \notin \boldsymbol{y}
\end{array}$$

On peut étendre cette arithmétique aux fonctions élémentaires (cos,sin,log,exp,...) [95]. D'autres opérateurs comme la puissance on été définis :

$$\boldsymbol{x}^n = \begin{cases} [1, 1] & \text{si } n = 0 \\ [\underline{x}^n, \overline{x}^n] & \text{si } \underline{x} \geq 0 \text{ ou si } 0 \in \boldsymbol{x} \text{ et } n \text{ est impair} \\ [\overline{x}^n, \underline{x}^n] & \text{si } \overline{x} \leq 0 \\ [0, \max(\underline{x}^n, \overline{x}^n)] & \text{si } 0 \in \boldsymbol{x} \text{ et } n \text{ est pair} \end{cases}$$

[1] Un récapitulatifs des notations utilisées dans ce manuscrits est présenté à la page x

Remarque L'addition et la multiplication sont commutatives et associatives :

$$\begin{array}{rcll} \text{Commutatitivité} & : & \boldsymbol{x} \oplus \boldsymbol{y} = \boldsymbol{y} \oplus \boldsymbol{x} & \boldsymbol{x} \otimes \boldsymbol{y} = \boldsymbol{y} \otimes \boldsymbol{x} \\ \text{Associativité} & : & \boldsymbol{x} \oplus (\boldsymbol{y} \oplus \boldsymbol{z}) = (\boldsymbol{x} \oplus \boldsymbol{y}) \oplus \boldsymbol{z} & \boldsymbol{x} \otimes (\boldsymbol{y} \otimes \boldsymbol{z}) = (\boldsymbol{x} \otimes \boldsymbol{y}) \otimes \boldsymbol{z} \end{array}$$

Cependant, la distributivité n'est plus valide. De manière générale, une propriété plus faible est vérifiée : la sous-distributivité [92]. Cette propriété énonce que quels que soient les intervalles $\boldsymbol{x}$, $\boldsymbol{y}$ et $\boldsymbol{z}$:

$$\text{Sous-distributivité :} \quad \boldsymbol{x} \otimes (\boldsymbol{y} \oplus \boldsymbol{z}) \subseteq \boldsymbol{x} \otimes \boldsymbol{y} \oplus \boldsymbol{x} \otimes \boldsymbol{z}$$

L'utilisation des nombres à virgule flottante pour les calculs d'expressions numériques entraîne des erreurs de calcul auxquelles l'arithmétique des intervalles peut pallier. Il suffit d'arrondir vers l'extérieur chaque opération de base [1], afin de garantir que le résultat réel soit inclus dans le résultat intervalle.

Plus précisément, étant donné un nombre flottant $f \in \mathbb{F}$, on désignera par f^+ le plus petit nombre flottant supérieur à f. Par analogie, f^- le plus grand nombre flottant inférieur à f. L'arrondi supérieur d'un réel r est le plus petit flottant supérieur à r, noté $\lceil r \rceil$. L'arrondi inférieur de r est le plus grand flottant inférieur à r, noté $\lfloor r \rfloor$. Un intervalle réel $[a, b] \in \mathbb{IR}$ est arrondi à l'extérieur par l'intervalle flottant $[\lfloor a \rfloor, \lceil b \rceil]$. Chaque intervalle résultant d'un calcul intermédiaire lors d'une évaluation par intervalles doit être arrondi de cette manière. Cela alourdi le calcul mais permet de garantir la *correction* du résultat.

Arithmétique étendue Dans les règles de calcul énoncées dans la définition 3.3.1, la division par un intervalle contenant zéro a été exclue. En effet, cela suppose de gérer correctement les valeurs $\pm\infty$. Une arithmétique étendue a donc été définie dans ce sens [65], dans laquelle les définitions des opérateurs de base $\oplus$, $\ominus$ et $\otimes$ sont notamment étendus aux valeurs $\pm\infty$. D'autre part, il est possible en utilisant l'arithmétique des intervalles étendus de calculer une évaluation de $\boldsymbol{x} \oslash \boldsymbol{y}$ lorsque $0 \in \boldsymbol{y}$ (Ex. 3.3.1). Cette évaluation est dans certains cas une union d'intervalles, comme le montre l'exemple suivant :

Exemple 3.3.1 *Soient* $\boldsymbol{x} = [-1, 1]$ *et* $\boldsymbol{y} = [-2, 2]$ *deux intervalles. L'évaluation par intervalles de* $\boldsymbol{x} \oslash \boldsymbol{y}$ *en utilisant l'arithmétique étendue est* $[-\infty, -1/2] \cup [1/2, \infty]$.

3.3.3 Extension aux intervalles

Les fonctions et contraintes numériques, tout comme les expressions arithmétiques, peuvent être étendues aux intervalles. De la même manière que pour les opérateurs arithmétiques, une extension aux intervalles d'une fonction f doit respecter certaines propriétés. On dira que $\boldsymbol{f}$ est une extension aux intervalles de f si $\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ est un intervalle contenant toutes les valeurs de f, lorsque les x_i prennent leurs valeurs dans $\boldsymbol{x}_i$. Formellement,

Définition 3.3.2 (Extension d'une fonction aux intervalles [92, 95]) *Soit* $f : \mathbb{R}^n \to \mathbb{R}$ *une fonction à valeurs réelles à* n *inconnues* $X = (x_1, \ldots, x_n)$. *Une* ***extension aux intervalles*** *de* f *est une fonction* $\boldsymbol{f} : \mathbb{IR}^n \to \mathbb{IR}$ *telle que :*

$$\forall \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{I} : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n \Rightarrow f(x_1, \ldots, x_n) \in \boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$$

Une évaluation par intervalle d'une fonction $f : \mathbb{R}^n \to \mathbb{R}$ sur une boite donnée $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ est un intervalle $\boldsymbol{y}$ tel que :

$$\forall x_i \in \boldsymbol{x}_i, 1 \leq i \leq n : \qquad \underline{y} \leq f(x_1, \ldots, x_n) \leq \overline{y}$$

En d'autres termes, $\boldsymbol{y}$ est un intervalle qui contient toutes les valeurs de f lorsque les valeurs des inconnues sont restreintes à la boîte $\boldsymbol{X}$.

La manière la plus simple de calculer $\boldsymbol{y}$ est d'évaluer l'**extension naturelle** de f (Ex. 3.3.2), obtenue en substituant tous les opérateurs mathématiques classiques (resp. les constantes et les variables) de f par leur extension *basique* aux intervalles [92, 95]. Le même principe peut être appliqué pour calculer une union d'intervalles qui contient toutes les valeurs de f, en utilisant l'arithmétique des intervalles étendue.

Exemple 3.3.2 (Extension naturelle) *Soit $f : \mathbb{R}^3 \to \mathbb{R}$ la fonction définie par $f(x, y, z) = 2x + yz - 1$. L'extension naturelle de f est la fonction intervalles $\boldsymbol{f} : \mathbb{IR}^3 \to \mathbb{IR}$ définie par $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = 2 \otimes \boldsymbol{x} \oplus \boldsymbol{y} \otimes \boldsymbol{z} \ominus 1$. Une évaluation de f sur la boîte $\boldsymbol{X} = ([-1, 1], [-1, 1], [1, 2])$ est donc $2 \otimes [-1, 1] \oplus [-1, 1] \otimes [1, 2] \ominus 1 = [-5, 3]$.*

La définition (Déf. 3.3.2) peut être étendue aux contraintes de la manière suivante :

Définition 3.3.3 (Extension d'une contrainte aux intervalles [28]) *Une **extension aux intervalles** d'une contrainte $c \subseteq \mathbb{R}^n$ est une relation $\boldsymbol{c} \subseteq \mathbb{IR}^n$ telle que :*

$$\forall \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{I} : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n \Rightarrow ((x_1, \ldots, x_n) \in c \Rightarrow (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \boldsymbol{c})$$

Une relation sur $\mathbb{R}$ peut être approximée par un intervalle contenant l'ensemble des éléments de cette relation (Déf. 3.3.4).

Définition 3.3.4 (Enveloppe intervalle d'une relation sur $\mathbb{R}$) *. Soit $\rho \subseteq \mathbb{R}$, un sous-ensemble de l'ensemble des réels, l'**enveloppe intervalle** de ρ, notée $\square_{\mathbb{I}}(\rho)$, est définie par l'intervalle :*

$$\square_{\mathbb{I}}(\rho) = [\inf \rho, \sup \rho]$$

Remarque On utilise ici la notation $\square_{\mathbb{I}}()$, pour différencier l'enveloppe intervalle de l'enveloppe union d'intervalles, notée $\square_{\mathbb{U}}()$, sur laquelle on reviendra dans le chapitre 8.

De même, une relation sur $\mathbb{R}^n$ peut être approximée par une boîte contenant l'ensemble des éléments de cette relation (Déf. 3.3.6). Cette boîte est le produit cartésien des enveloppes intervalles des projections de la relation sur chacune des dimension de $\mathbb{R}^n$ (Déf. 3.3.5).

Définition 3.3.5 (Projection d'une relation sur $\mathbb{R}^n$) *Soit $\rho \subseteq \mathbb{R}^n$, la **projection** de ρ sur la dimension i est définie par le sous-ensemble de $\mathbb{R}$:*

$$\pi_i(\rho) = \{v_i \in \mathbb{R} : (v_1, \ldots, v_i, \ldots, v_n) \in \rho\}$$

Définition 3.3.6 (Enveloppe intervalle d'une relation sur $\mathbb{R}^n$) *. Soit $\rho \subseteq \mathbb{R}^n$, l'**enveloppe intervalle** de ρ est définie par la boîte :*

$$(\square_{\mathbb{I}}(\pi_1(\rho)), \ldots, \square_{\mathbb{I}}(\pi_n(\rho))) \in \mathbb{IR}$$

3.3.4 Problème de dépendance des variables

Supposons que l'on veuille soustraire un intervalle $\boldsymbol{x} = [\underline{x}, \overline{x}]$ de lui-même. Le résultat attendu est évidemment 0 et pourtant en utilisant les règles de calcul énoncées plus haut on obtiendrait $[\underline{x}-\overline{x}, \overline{x}-\underline{x}]$. Le résultat obtenu est $\{x-y : x \in \boldsymbol{x}, y \in \boldsymbol{x}\}$ au lieu de $\{x-x : x \in \boldsymbol{x}\}$.

En fait, lorsqu'une variable apparaît plus d'une fois dans une expression (*occurrence multiple*) elle est traitée comme si c'était une variable différente à chaque occurrence. Dans l'exemple précédent, $\boldsymbol{x} - \boldsymbol{x}$ a été traité comme $\boldsymbol{x} - \boldsymbol{y}$ avec $\boldsymbol{x}$ égal à $\boldsymbol{y}$, comme si x et y étaient deux variables indépendantes.

Le problème de dépendance a pour effet d'augmenter la taille de l'intervalle résultant de l'évaluation d'une expression. Par conséquent, la forme syntaxique des fonctions joue un rôle déterminant dans leur évaluation par intervalles (Ex. 3.3.3) Il a été montré [92] qu'une expression sans occurrences multiples de la même variable s'évalue de manière exacte (Prop. 3.3.1).

Exemple 3.3.3 (Influence de la forme syntaxique des fonctions) *Considérons les fonctions $f_1(x, y) = x^2 + 2xy + y^2$ et $f_2(x, y) = (x + y)^2$, avec $x, y \in \boldsymbol{X} = (\boldsymbol{x}, \boldsymbol{y})$. Pour certaines valeurs de $\boldsymbol{x}$ et $\boldsymbol{y}$, l'évaluation de f_1 sur $\boldsymbol{X}$ est plus grossière que celle de f_2. Par exemple, si $\boldsymbol{X} = ([-1, 2], [-1, 2])$ alors $\boldsymbol{f_1}(\boldsymbol{X}) = [-4, 16]$ et $\boldsymbol{f_2}(\boldsymbol{X}) = [0, 16]$.*

Proposition 3.3.1 ([92]) *Soit $\boldsymbol{f} : \mathbb{IR}^n \to \mathbb{IR}$ l'extension naturelle aux intervalles de $f : \mathbb{R}^n \to \mathbb{R}$. Si toutes les variables n'apparaissent qu'une seule fois dans l'expression arithmétique de f alors :*

$$\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \square_{\mathbb{I}}\{f(v_1, \ldots, v_n) : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n\}$$

Sinon,

$$\boldsymbol{f}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \supseteq \square_{\mathbb{I}}\{f(v_1, \ldots, v_n) : x_1 \in \boldsymbol{x}_1, \ldots, x_n \in \boldsymbol{x}_n\}$$

La section suivante présente les algorithmes de résolution de CSPs continus basés sur les techniques de consistance.

3.4 Résolution de CSPs continus

La méthode la plus courante pour résoudre un CSP à domaine continu est d'utiliser un algorithme de type Branch&Prune, dont le schéma général est donné par la figure 3.1. Cet algorithme alterne réduction des domaines et décomposition de domaines jusqu'à ce qu'une précision ω sur les boîtes soit atteinte. La fonction Prune (ligne 5) est un des algorithmes de filtrage standard basé sur les techniques de consistances pour les CSPs numériques : Hull-consistance, Box-consistance ou kB-consistances. La fonction Split (ligne 10) sélectionne une variable et décompose le domaine associé. Les problèmes générés sont ajoutés à l'ensemble Q.

Dans la suite de cette section, on s'attachera à décrire plus précisément l'étape de splitting. Dans la section 3.5, les principales techniques de filtrage issues de la programmation par contrainte seront présentées.

La section suivante présente les différentes heuristiques pour améliorer la recherche des solutions.

```
BranchAndPrune(in :X_0, C, ω out : S)
%% X_0 = (x_1, ..., x_n)
Q ← {X_0}
S ← ∅
while Q ≠ ∅ do
  Extract X from Q
  X ← Prune(C, X)
  if X ≠ ∅ then
    if w(X) ≤ ω then
      S ← S ∪ X
    else
        % Standard splitting process
      Q ← Q∪ Split(X)
    endif
  endif
endwhile
return S
```

FIG. 3.1 – Schéma général de l'algorithme de recherche standard

3.4.1 Heuristiques de recherche

Tout comme pour l'algorithme de backtracking pour les CSPs finis, les performances d'un algorithme de recherche basé sur ce schéma dépend de plusieurs facteurs, dont l'ordre de sélection des domaines et le choix des points de coupe dans le domaine sélectionné.

Stratégies de sélection de domaine Étant donnée une boîte $\boldsymbol{X}$, le splitting consiste tout d'abord à choisir un domaine $\boldsymbol{x}_i$ à décomposer. Ce choix peut être déterminé de manière statique ou bien dynamiquement pendant la résolution. Les trois heuristiques les plus utilisées sont les suivantes :

- RR (Round Robin) : les domaines sont sélectionnés les uns à la suite des autres dans un ordre statique prédéterminé. Cette stratégie est considérée comme étant la plus efficace car elle permet d'équilibrer les choix de variables durant le processus de recherche.
- LF (Largest First) : à chaque étape de splitting, le domaine le plus grand est sélectionné. Cette stratégie est souvent combinée avec RR, car elle conduit souvent à sélectionner la même variable plusieurs fois d'affilée. Autrement dit, si une variable x_k est sélectionnée plus de p fois à la suite, un RR est alors utilisé pour passer à la variable x_{k+1}.
- MS (Maximal Smear) : à chaque étape de splitting, le domaine qui maximise la "smear function" est sélectionné. La smear function donne une indication sur le potentiel de filtrage d'une variable ; MS sélectionne la variable dont la projection sur l'une des contraintes a la plus grande pente. Plus précisément, la smear function de la variable x_k est définie par :

$$s_k = \max_{1 \leq j \leq m} \{\max\{|\underline{J}_{i,j}|, |\overline{J}_{i,j}|\} w(\boldsymbol{x}_i)\},$$

  où $\boldsymbol{J}_{i,j} = [\underline{J}_{i,j}, \overline{J}_{i,j}]$ est la (i,j)-ème entrée de l'extension aux intervalles de la matrice Jacobienne du système.

Stratégies de décomposition Une fois que le domaine d'une variable a été sélectionné, il s'agit de déterminer un ou plusieurs points de coupe à l'intérieur de ce domaine pour générer les sous-problèmes.

La stratégie à la fois la plus simple et la plus efficace en pratique est de couper en 2 parties de même taille : la *bisection* ; c'est-à-dire le point de coupe choisi est le milieu de l'intervalle. Concernant la bisection, il n'est pas nécessaire de couper l'intervalle choisi en son milieu, on peut choisir n'importe quel point de l'intervalle. Quelques études ont porté sur le choix de ce point de coupe notamment dans [58], mais sans toutefois proposer de résultats expérimentaux.

D'autres stratégies de splitting sélectionnent plusieurs points de coupe en général dans le même domaine [95, 66, 67, 58]. Par exemple, la multisection découpe le domaine sélectionné en k parties de même taille. En pratique, il est souvent plus avantageux de couper le domaine en 2 ou 3 parties puis de faire confiance au filtrage pour réduire l'espace de recherche.

La section suivante présente les principales techniques de filtrage par consistance pour les CSPs continus.

3.5 Consistances sur les CSPs continus

Le filtrage joue un rôle important dans la recherche de solution car il permet de réduire l'espace de recherche. Différents algorithmes de réduction des bornes des domaines ont été conçus et définis par des propriétés de consistances. La principale caractéristique de ces propriétés est leur complétude, c'est-à-dire qu'elles garantissent qu'aucune solution n'est perdue. En revanche, elles ne garantissent pas l'existence de solutions (pas davantage que les filtrages sur les domaines finis).

Tout comme pour les CSPs finis, il y a différents niveaux de consistance pour les CSPs continus qui se divisent en deux catégories : les consistances locales et les consistances partielles. Les consistances locales sont des propriétés qui ne portent que sur une seule contrainte. Les consistances partielles mettent en jeu un ensemble de contraintes. Ces propriétés sont établies par des algorithmes qui éliminent des domaines les valeurs inconsistantes. Pour éviter l'explosion combinatoire due à la gestion d'unions d'intervalles, les réductions des domaine sont en général opérées uniquement aux bornes des domaines. [53] a toutefois proposé un algorithme pour calculer la consistance d'union en combinant des ensembles d'intervalles. Nous verrons dans le chapitre 8, comment utiliser ces union d'intervalles pour améliorer les performances de la recherche de solutions..

Les sections 3.5.1 et 3.5.2 décrivent respectivement les principales techniques de filtrage par consistance locale et par consistance partielle.

3.5.1 Consistances locales

L'algorithme classique pour calculer un filtrage par consistance locale, nommé IN-1 (Fig. 3.2), est basé sur le même schéma que l'algorithme AC-3 (Fig. 2.5 page 17) qui calcule l'arc-consistance pour des CSPs finis. Cet algorithme utilise une procédure de réduction des domaines, nommée Narrow, qui joue le même rôle que la procédure Revise (Fig. 2.4 page 16).

Autrement dit, Narrow établit une propriété de consistance locale Φ sur les domaines des variables d'une contrainte, en éliminant les valeurs ne respectant pas Φ. Cette propriété de consistance est associée à un opérateur de narrowing (Déf. 3.5.1), qui garantit la convergence de IN-1 et la conservation de l'espace des solutions.

```
      IN-1(in-out :P = (𝒳,𝒟,𝒞))
1 :   Q ← {⟨x_i, c_j⟩ : x_i ∈ 𝒳(c_j)}
2 :   while Q ≠ ∅ do
3 :     extraire ⟨x_i, c⟩ de Q
6 :     if Narrow(x_i,c_j) then
7 :       Q ← Q ∪ {⟨x_k, c⟩ : c ≠ c_j ∧ x_i, x_k ∈ 𝒳(c)}
8 :     endif
9 :   endwhile
10 :  return
```

FIG. 3.2 – Algorithme de propagation IN-1

Définition 3.5.1 (Opérateurs de narrowing [96]) *Soit c une contrainte n-aire. Un* ***opérateur de narrowing*** *pour la contrainte c est une application $\phi : \mathbb{IR}^n \to \mathbb{IR}^n$ telle que pour tout $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{IR}$ les propriétés suivantes sont vérifiées :*

$$\begin{array}{cc} \phi(\boldsymbol{X}) \subseteq \boldsymbol{X} & \textit{(Contractance)} \\ c \cap \boldsymbol{X} \subseteq \phi(\boldsymbol{X}) & \textit{(Correction)} \\ \boldsymbol{X} \subseteq \boldsymbol{Y} \Longrightarrow \phi(\boldsymbol{X}) \subseteq \phi(\boldsymbol{Y}) & \textit{(Monotonie)} \end{array}$$

Le filtrage par propagation peut engendrer des cycles de réductions faibles et de ce fait entraîner des phénomènes de convergence lente. Cela se produit en particulier quand la fonction Narrow réduit faiblement les domaines. Différentes techniques d'accélération de la convergence ont été proposées, comme les méthodes d'extrapolation [74] et la détection de cycles de propagation [78]. La propagation peut également être court-circuitée lorsque la réduction du domaine n'est pas suffisante.

Les deux consistances locales les plus utilisées sont la Hull-consistance ou 2B-consistance [79, 80, 15, 13, 27, 28] et la Box-consistance [14, 119, 26]. Les deux paragraphes suivants décrivent ces consistances locales.

Hull-consistance La Hull-consistance ou 2B-consistance est une approximation de l'arc-consistance. L'arc-consistance établit une propriété sur tous les éléments du domaine, alors que la Hull-consistance établit cette propriété uniquement sur les bornes. Une contrainte c est dite Hull-consistante si pour toute variable x_i de $\mathcal{X}(c)$, il existe une valeur dans les domaines des autres variables qui satisfait c lorsque x_i est instanciée à $\underline{x}_i$ ou à $\overline{x}_i$.

Le calcul de la Hull-consistance est basée sur la notion de projection de contrainte (Déf. 3.5.2) ou fonction de projection. La projection d'une contrainte c sur une variable x_k pour une boîte $\boldsymbol{X}$ est l'ensemble des valeurs de $v_k \in \boldsymbol{x}_k$ pour lesquelles il existe une affectation des autres variables dans leurs domaines qui satisfait c. Plus formellement,

Définition 3.5.2 (Projection d'une contrainte) *Soit $c(x_1, \ldots, x_n)$ une contrainte n-aire. La k-ème projection de c par rapport à la boîte $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ est définie par :*

$$\begin{array}{ll} \pi_k(c \cap \boldsymbol{X}) = \{v_k \in \boldsymbol{x}_k | & \exists v_1 \in \boldsymbol{x}_1, \ldots, \exists v_{k-1} \in \boldsymbol{x}_{k-1}, \\ & \exists v_{k+1} \in \boldsymbol{x}_{k+1}, \ldots, \exists v_n \in \boldsymbol{x}_n \quad t.q.\ (v_1, \ldots, v_{k-1}, v_k, v_{k+1}, v_n) \in c\} \end{array}$$

Ainsi, une contrainte c est Hull-consistante si le domaine de chacune de ses variables x_k est égal à l'intervalle englobant la projection de c sur x_k :

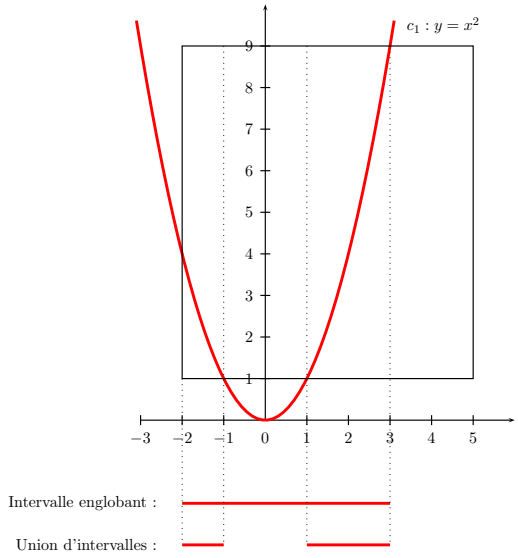


FIG. 3.3 – Projection d'une contrainte

Définition 3.5.3 (Hull-consistance [79, 15]) *Soit c une contrainte n-aire. c est Hull-consistante sur la boîte $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ ssi $\forall x_k \in \mathcal{X}(c), \boldsymbol{x}_k = \square_{\mathbb{I}}(\pi_k(c \cap \boldsymbol{X}))$. Un CSP est Hull-consistant ssi toutes ses contraintes sont Hull-consistantes.*

Il n'est en général pas possible de calculer une fonction de projection de manière exacte.Par contre, ces fonctions de projections peuvent être approximées par un intervalle englobant, en général c'est ce que l'on obtient en utilisant l'arithmétique de base sur les intervalles. En revanche, en utilisant l'arithmétique étendue, on peut approximer ces fonctions de projection de manière plus fine par une union d'intervalles. Par exemple, la figure 3.3 montre que la projection sur x de la contrainte $c : y = x^2$ pour la boîte $[-3, 3] \times [1, 9]$ peut être approximée par l'intervalle $[-3, 3]$ ou l'union d'intervalles $[-3, -1] \cup [1, 3]$.

L'évaluation de ces fonctions de projections est donc au coeur des algorithmes qui calculent la Hull-consistance. Différentes techniques ont été proposées pour calculer ces fonctions de projection. La première de ces techniques consiste à décomposer le système de contraintes en contraintes élémentaires (unaires, binaires ou ternaires) pour lesquelles il est facile d'évaluer ces projections. Par exemple, la décomposition en contraintes élémentaire de la contrainte $c : (xy)^3 + z^2 + xy = 0$ est le système suivant :

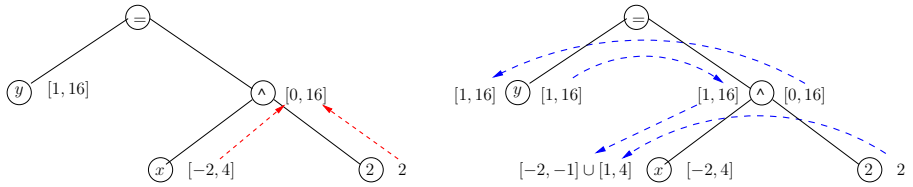
$$\begin{cases} X^3 + Y + X & = & 0, \\ X & = & xy, \\ Y & = & z^2, \end{cases}$$

L'inconvénient d'une telle décomposition est l'introduction de nombreuses variables additionnelles, qui affaiblissent en général le filtrage. Par ailleutrs, celles-ci introduisent un coût supplémentaire à la fois en mémoire et en temps de calcul.

L'implantation la plus efficace de la Hull-consistance, nommée HC4 [13], est basée sur l'opérateur de réduction HC4Revise. La figure 3.4 illustre l'exécution de HC4Revise sur la contrainte $c : y = x^2$, lorsque le vecteur (x, y) varie dans la boîte $[-2, 4] \times [1, 16]$.

Cette implantation ne nécessite aucune décomposition explicite des contraintes. Toutes les projections sont calculées en traversant la représentation arborescente des contraintes en deux étapes :

- *Forward Propagation* : Durant cette phase, les expressions associées à chaque noeud sont évaluées par un parcours infixe de l'arbre syntaxique. Les valeurs associées à chaque noeud de l'arbre sont calculées en utilisant l'opérateur correspondant et les domaines associés à ses sous-arbres.
- *Backward Propagation* : L'arbre est parcouru dans le sens inverse pour réduire les domaines des variables, en utilisant les fonctions de projection associées à chacun des opérateurs.



(a) Forward propagation (b) Backward propagation

FIG. 3.4 – Mise en œuvre de HC4revise sur la contrainte $y = x^2$ avec $(x, y) \in [-2, 4] \times [1, 16]$.

Box-consistance La Box-consistance [14, 119] est une approximation plus grossière de l'arc-consistance que la Hull-consistance. Pourtant, en pratique, elle permet d'obtenir un filtrage plus fort des domaines car elle permet dans certains cas de remédier au problème de dépendances des variables.

Une contrainte c est Box-consistante si pour toutes les variables x_i de $\mathbb{V}_c$, les bornes de $\boldsymbol{x}_i$ satisfont la contrainte unaire obtenue en remplaçant chaque occurrence d'une variable x_j autre que x_i par l'intervalle constant $\boldsymbol{x}_j$. Plus formellement,

Définition 3.5.4 (Box-consistance [14, 119]) *Soit c une contrainte n-aire. c est Box-consistante pour la boîte $\boldsymbol{X}$ ssi $\forall x_i$ les deux propriétés suivantes sont vérifiées :*

1. $\boldsymbol{c}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, [\underline{x}_i, \underline{x}_i^+), \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)$
2. $\boldsymbol{c}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, (\overline{x}_i^-, \overline{x}_i], \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)$

Un CSP est Box-consistant si toutes ses contraintes sont Box-consistantes.

La Box-consistance génère un ensemble de fonctions univariées qui peuvent être résolues par des méthodes numériques comme la méthode de Newton [49]. Le filtrage consiste à trouver les quasi-zéros extrêmes de ces fonctions univariées. Un quasi-zéro est défini de la manière suivante :

Définition 3.5.5 (Quasi-Zéro) *Un quasi-zéro de la fonction $\boldsymbol{f}$ est une boîte $\boldsymbol{X}$ telle que $0 \in \boldsymbol{f}(\boldsymbol{X})$.*

Soit BCNarrow, l'opérateur de réduction associé à la Box-consistance. BCNarrow(c,$\boldsymbol{X}$) réduit le domaine de chaque variables de c jusqu'à ce que c soit Box-consistante. Pour chaque x_k de c, une fonction intervalle univariée $\boldsymbol{f}_{\boldsymbol{x}_k}$ est générée à partir de c, en remplaçant toutes les

```
BCNarrow(in :c, X) : Interval vector
% X = (x_1, ..., x_n)
foreach x_k ∈ V_c do
   x_k ← LeftNarrow(f_{x_k}, f'_{x_k}, x_k)
   x_k ← RightNarrow(f_{x_k}, f'_{x_k}, x_k)
   if x_k = ∅ then
     return ∅
   endif
endfor
return X
```

FIG. 3.5 – L'opérateur de Narrowing associé à la Box-consistance

variables sauf x_k par leur domaine associé. Le filtrage consiste alors à trouver le quasi-zéro le plus à gauche et le quasi-zéro le plus à droite de $\boldsymbol{f}_{\boldsymbol{x}_k}$.

Cette réduction est calculée pour la borne inférieure par la fonction LeftNarrow (voir figure 3.6) et pour la borne supérieure par la fonction RightNarrow (voir [14, 119] pour plus de détails sur ces algorithmes).

```
LeftNarrow(in :f,f',x) : Interval
r ← x̄
if 0 ∉ f(x) then
   return ∅
endif
x ← MonoNewton(f, f', x)
if 0 ∈ f([x̲, x̲⁺]) then
   return [x̲, r]
else
   l ← LeftNarrow(f, f', [x̲, m(x)])
   if l = ∅ then
     l ← LeftNarrow(f, f', [m(x), x̄])
   endif
   return [l̲, r]
endif
return x
```

FIG. 3.6 – LeftNarrow :L'opérateur de réduction de la borne inférieure, qui recherche le quasi-zéro le plus à droite.

De manière informelle, LeftNarrow tente de réduire la borne inférieure d'un domaine $\boldsymbol{x} = [a, b]$ par rapport à une contrainte c. Pour cela, LeftNarrow vérifie si $[a, \frac{a+b}{2}]$ est un quasi-zéro de $\boldsymbol{f}_x$. Si ce n'est pas le cas alors la recherche du quasi-zéro le plus à droite se poursuit avec l'intervalle $[\frac{a+b}{2}, b]$. Sinon, LeftNarrow réduit $[a, \frac{a+b}{2}]$ avec la méthode de Newton monovarié et poursuit sa recherche dans la moitié gauche de l'intervalle réduit. L'opérateur de Newton monovarié sera détaillé dans la section 8.3.3 page 94.

La section suivante présente les techniques de consistance partielles pour les CSPs conti-

```
3BPrune(in-out : P)
1 :  FP = (fp_1, ..., fp_n)
2 :  S = (s_1, ..., s_n)
3 :  foreach i ≤ n do
4 :    fp_i ← false
5 :    s_i ← +∞
6 :  endfor
7 :  while ∃i s.t. fp_i = false do
8 :    foreach i ≤ n do
9 :      if ¬fp_i then
10 :       s_i ← min(s_i, w(x_i))/2
11 :       if s_i ≤ ω then
12 :         fp_i ← true
13 :         s_i ← ω
14 :       endif
15 :     endif
16 :   endfor
17 :   X = 3BFixpoint(P, S)
18 : endwhile
```

```
3BFixpoint(in : P, S)
1 :  fp ← false
2 :  foreach i ≤ n do
3 :    if 2BPrune(P_{x_i ← [x̲_i, x̲_i + s_i]}) = ∅ then
4 :      fp ← false
5 :      x_i ← x_i \ [x̲_i, x̲_i + s_i]
6 :      X ← 2BPrune(P)
7 :    endif
8 :    if 2BPrune(P_{x_i ← [x̄_i − s_i, x̄_i]}) = ∅ then
9 :      fp ← false
10 :     x_i ← x_i \ [x̄_i − s_i, x̄_i]
11 :     X ← 2BPrune(P)
12 :   endif
13 : endfor
14 : return X
```

FIG. 3.7 – 3BPrune : Algorithme de fermeture par 3B-consistance

nus.

3.5.2 Consistances partielles

Les consistances partielles, comme les kB-consistances [79, 74], sont des consistances qui ne sont pas strictement locales. Pour réduire les domaines, ces consistances tentent de prouver que certaines parties de ces domaines sont inconsistantes. La kB-consistance tente de rejeter $[\underline{x}, s]$ ou $[s, \overline{x}]$, avec $s \in [\underline{x}, \overline{x}]$, en utilisant une consistance plus faible, la $(k-1)$B-consistance.

La kB-consistance est a une définition récursive basée sur la 2B-consistance, qui est équivalente à la Hull-consistance. D'autres consistances peuvent être définie en se basant sur d'autres consistances locales. C'est le cas de la Bound-consistance [100], qui est similaire à la 3B-consistance mais avec un schéma récursif basé sur la Box-consistance. Plus formellement,

Définition 3.5.6 (kB-consistance) *Soit P un CSP, on note $P_{\boldsymbol{x}_i \leftarrow \boldsymbol{\alpha}}$, le CSP dérivé de P en substituant $\boldsymbol{x}_i$ par l'intervalle $\boldsymbol{\alpha}$.*
*P est dit k**B-consistant** ssi il vérifie les propriétés suivantes :*
- $P_{\boldsymbol{x}_i \leftarrow [\underline{x}_i, \underline{x}_i^+)}$ *est $(k-1)$B-consistant.*
- $P_{\boldsymbol{x}_i \leftarrow (\overline{x}_i^-, \overline{x}_i]}$ *est $(k-1)$B-consistant.*

L'algorithme de fermeture par 3B-consistance[2] est donné par la figure 3.7. Cet algorithme, que nous nommerons 3BPrune, n'est pas basé sur le schéma standard de propagation par opposition aux consistances locales (Hull et Box-consistance).

[2]Cet algorithme s'étend trivialement au calcul de la kB-consistance

Le principe de 3BPrune est de réfuter certaines portions du domaine d'une variable sans créer de point de choix. Autrement dit, si le domaine d'une variable d'un CSP P est $\boldsymbol{x} = [a, b]$, 3BPrune va tenter d'éliminer la portion $[a, \frac{a+b}{2})$ en appliquant un filtrage par 2B-consistance sur le CSP $P_{\boldsymbol{x} \leftarrow [a, \frac{a+b}{2})}$. Si la 2B-consistance prouve que l'intervalle $[a, \frac{a+b}{2})$ est inconsistant alors le filtrage se poursuit avec l'intervalle $[\frac{a+b}{2}, b]$, sinon 3BPrune va essayer d'éliminer l'intervalle $[a, \frac{a+b}{4})$. L'algorithme s'arrête lorsque le point fixe est atteint ou que les réductions calculées sont inférieures à un ω donné.

Les kB-consistances sont très peu utilisées dans la mesure où les algorithmes pour les calculer sont très coûteux en temps de calcul. En pratique, la 3B-consistance est un bon compromis entre qualité et efficacité de filtrage pour des problèmes difficiles. La Bound-consistance a permis de résoudre efficacement plusieurs problèmes difficiles comme le problème du transistor [99, 100]. Les performances de la 4B-consistance sur ce problème sont également significatives [74].

La section suivante présente les principales contraintes globales sur les domaines continus.

3.6 Contraintes globales

Tout comme pour les CSPs finis, il est d'un grand intérêt de trouver des algorithmes de filtrage à la fois efficaces et puissants pour résoudre des ensembles de contraintes d'un même type. Toutefois, les contraintes globales en domaines continus reste un domaine dans lequel très peu de recherches ont été entreprises.

Certaines méthodes de résolution issues de l'analyse numérique ou l'analyse par intervalles peuvent être assimilés à des contraintes globales : par exemple, la méthode du simplexe pour les systèmes linéaires, ou la méthode de Newton multivarié.

Plus récemment, des travaux ont été entrepris sur les contraintes globales pour des CSPs continus. Parmi ces travaux, on peut mentionner la contrainte IS [108], qui regroupe des contraintes de somme sur des domaines finis et d'inégalité linéaires, et la contrainte Quad [76, 89] qui regroupe des contraintes quadratiques. Cette contrainte a été étendue pour traiter tout type de contraintes [75].

La contrainte IS a été introduite pour résoudre différents types d'applications, essentiellement pour des problèmes d'ordonnancement. IS est définie par :

- Une contrainte de somme :

$$\mathsf{Sum} : y = \sum_{i=0}^{i=n} x_i$$

- Un ensemble d'inéquations binaires :

$$\mathsf{Ineq} : \{x_i - x_j \leq c_{ij} : i, j \leq n, i \neq j\}$$

- Un ensemble de contraintes de domaines :

$$\mathsf{Dom} : \{\underline{x}_i \leq x_i \leq \overline{x}_i, i \leq n\}$$

La contrainte globale est alors définie par :

$$\mathsf{IS} = \mathsf{Sum} \cup \mathsf{Ineq} \cup \mathsf{Dom}$$

L'algorithme de calcul de la consistance d'intervalles pour la contrainte IS est basé sur un calcul de plus court chemin pour réduire les bornes des domaines des variables. Cet algorithme a donc une complexité en temps de calcul de l'ordre de mn, où n est le nombre de variables et m le nombre de contraintes d'inégalités.

La contrainte globale Quad est un algorithme de filtrage global pour les systèmes de contraintes quadratiques (*c.f* [76, 89, 75] pour une description plus détaillée). Cet algorithme repose sur une approximation des contraintes quadratiques du système initial par un ensemble de contraintes linéaires. Le simplexe est utilisé pour calculer une borne inférieure et supérieure des variables.

La relaxation des contraintes quadratiques consiste d'abord à introduire une nouvelle variable pour chaque terme non linéaire des contraintes quadratiques. Les domaines des variables additionnelles sont calculés en utilisant l'arithmétique de base sur les intervalles. Les termes quadratiques sont ensuite approximés par deux classes de relaxations linéaires.

Ces relaxations génèrent des surestimateurs et des sous-estimateurs linéaires qui définissent une enveloppe convexe des termes quadratiques. Ces approximations apportent une meilleure estimation de l'ensemble des solutions que les fonctions de projection basées sur l'arithmétique des intervalles. Le point important est que ces relaxations linéaires sont "safe", autrement dit garantissent qu'aucune solution ne sera perdue durant le processus de filtrage.

Une fois le système de contraintes relaxé par un système linéaire, Quad fait appel à l'algorithme du simplexe pour réduire les bornes des intervalles. Plus précisemment, pour n variables, Quad fait $2n$ appels au simplexe pour minimiser et maximiser chacune des variables du système.

Du discret au continu Intuitivement, on pourrait envisager d'étendre les contraintes globales en domaines finis aux domaines continus. Très peu d'initiatives de cet ordre ont été entreprises. Il est en effet très difficile d'adapter les algorithmes de filtrages en domaines finis aux intervalles pour différentes raisons :

- La définition de ces contraintes n'est pas extensible à des domaines continus, car ils modélisent des problèmes purement finis. C'est le cas par exemple pour la contrainte Cycle, qui permet de trouver des cycles de longueur donnée dans un graphe orienté tels que chaque noeud soit visité une seule fois.
- Certaines contraintes globales utilisent une représentation des contraintes par un graphe dont le nombre de noeuds dépend du nombre de valeurs dans les domaines. C'est le cas par exemple pour la contrainte AllDiff, qui génère un graphe bi-parti dont les arêtes relient l'ensemble des variables à l'ensemble des valeurs (voir section 2.5), ou de manière similaire pour GCC, qui utilise des algorithmes de flots.
- Certaines contraintes globales ont une sémantique forte en domaines finis ; c'est justement sur cette sémantique que s'appuient les techniques de filtrage global. Or, pour un même contrainte, lorsque les variables varient dans des domaines continus, cette sémantique peut se perdre dès lors que les domaines des variables sont continus. Pour illustrer cette idée, prenons là encore la contrainte AllDiff sur un système à 3 variables x_1, x_2, x_3 et leurs domaines intervalles associés $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$. En supposant que ces intervalles ne soient pas dégénérés, il est toujours possible de trouver une affectation des variables par des valeurs différentes. Autrement dit, la consistance d'intervalle est toujours vérifiée.

3.7 Autres techniques de résolution

L'algorithme de Branch&Prune n'est plus adapté dès lors que l'espace des solutions est continu, ce qui est souvent le cas avec des systèmes d'inégalités. Cet algorithme génère dans

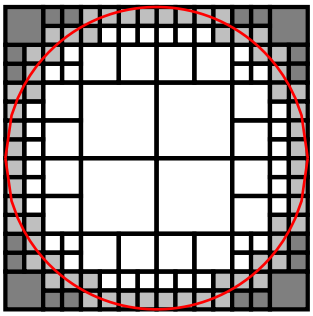


FIG. 3.8 – Décomposition en 2^k-arbre de la contrainte $x^2 + y^2 <= 1$, avec $\boldsymbol{x} = \boldsymbol{y} = [-1, 1]$: les pavés Inner sont représentées en blanc, les zones Outer en gris foncé et les zone Union en gris clair.

ce cas un nombre de boîtes solutions qui peut être très grand selon la précision des calculs.

Des techniques permettant d'approximer ces sous-espaces continus de solutions, ou *continuums de solutions*, ont été proposées. Sam-Haroud [113] a proposé une technique de pavage de l'espace en 3 types de régions : des pavés contenant uniquement des solutions (Inner), des pavés ne contenant aucune solution(Outer) et des pavés pour lesquels il est impossible de se prononcer (Union). Cette technique de décomposition, nommée décomposition en 2^k-arbres, permet une représentation fine par un pavage de l'espace des solutions (voir figure 3.8).

Deuxième partie

Contraintes globales pour les CSPs continus

Chapitre 4

Contraintes de distance et applications

Les systèmes de contraintes de distance sont présents dans un large champ d'applications (conception optimale de robots, contraintes géométriques, CAO, conformation moléculaire, ...). Ces systèmes sont définis par un ensemble de points et de distances entre certains couples de ces points. Le problème est de localiser ces points dans l'espace de manière à ce que les contraintes de distance soient respectées.

Les outils classiques pour résoudre les contraintes numériques sont basés sur des consistantes locales comme la 2B-consistance [79] ou la Box-consistance [14, 119]. L'inconvénient de ces méthodes vient du fait que les contraintes sont traitées indépendamment et sans exploiter leurs propriétés sémantiques. Par exemple, les variables associées aux coordonnées des points sont traitées comme des variables indépendantes.

Or, beaucoup d'informations peuvent être exploitées si l'on considère ces contraintes simultanément, c'est-à-dire comme une contrainte globale. Dans les chapitres 5 et 6, nous présenterons deux méthodes qui tentent de tirer parti de certaines de ces informations.

Dans ce chapitre, nous décrivons formellement le problème ainsi que les principales approches pour le modéliser et pour le résoudre.

4.1 Définitions et notations

On considère un ensemble de points $\mathcal{P}_i(x_{i_1}, \ldots, x_{i_p})$ de $\mathbb{R}^p$. La distance euclidienne δ_{ij} entre 2 points $\mathcal{P}_i$ et $\mathcal{P}_j$ est le nombre positif défini par la norme au carré du vecteur $\overrightarrow{\mathcal{P}_i\mathcal{P}_j}$:

$$\delta_{ij}^2 = \|\overrightarrow{\mathcal{P}_i\mathcal{P}_j}\|^2 = \sum_{k=1}^{k=p} (x_{i_k} - x_{j_k})^2 \tag{4.1}$$

Un problème de satisfaction de contraintes de distance est la donnée d'un certain nombre de distances entre des couples de points. Résoudre le problème c'est de localiser ces points dans l'espace de manière à ce qu'ils vérifient les distances données. Plus généralement, ces distances peuvent être données par un intervalle ; dans ce cas, on cherche à borner une zone de faisabilité pour chacun des points.

4.2 Modélisations

Il existe essentiellement deux modélisations différentes du problème. La première est la formulation usuelle que l'on utilisera le plus souvent dans la suite. Les inconnues sont les coordonnées des points dans un repère donné. Dans ce cas les contraintes sont de la même forme que dans l'équation 4.1. Plus formellement, un DCSP (Distance Constraint Satisfaction Problem) est défini par :

- Un ensemble de n points $\mathcal{P} = \{\mathcal{P}_i\}_{1 \leq i \leq n}$ dans un $\mathbb{R}^p$. On dénote par x_{i_k}, la k-ème coordonnée du point $\mathcal{P}_i$.
- Un ensemble de domaines $\mathcal{D} = \{\boldsymbol{\mathcal{P}}_i\}_{1 \leq i \leq n}$, où le vecteur d'intervalles $\boldsymbol{\mathcal{P}}_i = (\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_p})$ est le domaine du point $\mathcal{P}_i$.
- Un ensemble de contraintes de distance :

$$\mathcal{C} = \{c_{ij} : \sum_{k=1}^{k=p} (x_{ik} - x_{jk})^2 = \delta_{ij}^2\},$$

  où $1 \leq i, j \leq n$. La distance δ_{ij} est un nombre réel strictement positif ou un intervalle $[\underline{\delta_{ij}}, \overline{\delta_{ij}}]$.

La deuxième approche est de chercher des valeurs pour les distances inconnues. En effet, une fois que toutes les distances sont connues, le système devient trivial et indépendant du repère choisi. Cette modélisation est basée sur la géométrie des distances initiée dans les années 50 par Blumenthal [20]. Cette modélisation permet de représenter la plupart des contraintes géométriques (distance, incidence, orthogonalité, parallélisme ...) par des équations non-linéaires qui ne dépendent pas des coordonnées des points, mais des distances entre les points. Ces équations sont définies par des contraintes sur les déterminants de Cayley-Menger, définis de la manière suivante :

$$\Xi(\mathcal{P}_1, \ldots, \mathcal{P}_n) = \begin{vmatrix} 0 & \delta_{12}^2 & \delta_{13}^2 & \ldots & \delta_{1n}^2 & 1 \\ \delta_{21}^2 & 0 & \delta_{23}^2 & \ldots & \delta_{2n}^2 & 1 \\ \delta_{31}^2 & \delta_{32}^2 & 0 & \ldots & \delta_{3n}^2 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ \delta_{n1}^2 & \delta_{n2}^2 & \delta_{n3}^2 & \ldots & 0 & 1 \\ 1 & 1 & 1 & \ldots & 1 & 0 \end{vmatrix} \tag{4.2}$$

Par exemple, si $n = 2$: $\Xi(\mathcal{P}_1, \mathcal{P}_2) = 2\delta_{12}^2$. Si $n = 3$, $\Xi(\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3) = -16A^2$, où A est l'aire du triangle $\mathcal{P}_1\mathcal{P}_2\mathcal{P}_3$. Pour $n = 4$, une équation relie le déterminant de Cayley-Menger et le volume du tétraèdre $\mathcal{P}_1\mathcal{P}_2\mathcal{P}_3\mathcal{P}_4$. Pour $n > 4$, si ce déterminant est nul alors l'ensemble des points $\mathcal{P}_i$ est représentable dans $\mathbb{R}^3$. Plus généralement, en dimension p, si $n > p + 1$ et si $\Xi(\mathcal{P}_1, \ldots, \mathcal{P}_n) = 0$, alors il existe une configuration des points $\mathcal{P}_i$ dans l'espace $\mathbb{R}^p$ qui vérifie l'ensemble des contraintes de distance indiquées par la matrice de Cayley-Menger.

4.3 Applications

Les systèmes d'équations de distance euclidienne apparaissent dans de nombreux domaines d'application. Parmi ces applications, on détaille dans cette section la conception de robots, les contraintes géométriques, la conformation moléculaire, la localisation de capteurs mobiles et le circle packing. Des outils algorithmiques pour résoudre le problème pour chacune de ces applications ont été développés ; nous en donnons ici les grandes lignes sans entrer dans les détails.

Conception de robots [45, 72, 93, 37, 36, 87] : Un problème crucial en conception de robots est de résoudre le problème de la cinématique directe de robots. Un des exemples les plus étudiés pendant ces dernières années est la plateforme de Gough-Stewart. Ce robot parallèle est constitué d'une plateforme relié au sol par 6 jambes de longueurs fixées. Le but est de trouver toutes les positions possibles de la plateforme dans l'espace. Raghavan [102] et Ronga [109] ont été les premiers à montrer que ce problème avait au plus 40 solutions réelles et complexes. Husty [52] a exhibé un polynôme de degré 40 qui permet de déterminer toutes les solutions du système. Dietmeier [37] a exhibé des configurations de longueurs de jambes pour lesquelles il existe 40 solutions réelles. Ce problème peut être résolu en utilisant des méthodes d'élimination [55, 77], mais ces méthodes ne garantissent pas la correction des solutions. Des méthodes de continuation [81, 114, 102] permettent également de résoudre ce problème mais sont numériquement instables en présence de singularités. Les bases de Groëbner [72, 73, 56, 111] sont très efficaces et fournissent des résultats garantis. En revanche, les méthodes de résolution basées sur les bases de Groëbner sont exponentielles et donc très sensibles aux nombre de variables. Plus récemment, Merlet [87, 57] a montré qu'il était possible en utilisant des méthodes d'analyse par intervalles de fournir des résultats certifiés avec des temps de calcul très compétitifs avec les autres méthodes.

Conformation moléculaire [31, 39, 69, 70] : Ce problème est tout à fait similaire au problème précédent. Les points dont on cherche à déterminer la position correspondent aux atomes d'une molécule, les distances aux distances inter-atomiques mesurées la plupart du temps par des techniques de raisonnance magnétique. La conformation moléculaire consiste étant donnée une molécule à trouver des molécules ayant la même structure spatiale afin d'en mesurer les propriétés chimiques. Ce problème intéresse particulièrement les industries pharmaceutiques pour lesquelles l'enjeu est la fabrication de nouveaux médicaments. L'équivalence de ce problème avec celui de la cinématique directe des robots est illustrée par Emiris et Mourrain [39]. Krippahl et Barahona [69, 70] utilisent une modélisation discrète[1] du problème et utilisent des outils issus des CSPs finis pour le résoudre. Dans [31], Crippen et Havel décrivent des méthodes de résolution basées sur la deuxième modélisation (les inconnues sont les distances inter-points). La plupart de ces méthodes sont basées sur des algorithmes probabilistes. La modélisation par matrices de Cayley-Menger a été utilisée par Porta *et. al.* [97] pour résoudre ce type de problèmes. Cette méthode est une méthode basée sur le Branch&Prune avec un filtrage adapté pour réduire le domaine des distances. Ces domaines sont approximés par des polytopes convexes ; le filtrage est réalisé par un *clipping* du polytope, c'est-à-dire en intersectant ce polytope avec le demi-espace délimité par un plan.

Contraintes géométriques [20, 22, 19, 62, 60, 59, 61] : Les contraintes géométriques interviennent dans de nombreux domaines comme en CAO ou en biologie moléculaire. Ces problèmes consistent à trouver les positions, les orientations ou les dimensions d'objets géométriques soumis à des relations géométriques comme les distances mais aussi le parallélisme, l'orthogonalité ... Ces objets géométriques sont par exemple des points, des droites, des plans. Les outils classiques pour résoudre des CSPs géométriques sont très variés. Des techniques comme les bases de Gröebner, les méthodes d'élimination ou de continuation citées précédemment

[1] Les distances inter-atomiques étant très petites, on peut considérer que les positions des points dans l'espace sont limitées aux jonctions d'une grille. Les distances euclidiennes sont approximées par une distance sur chacun des axes de cette grille.

peuvent être utilisées. Des méthodes purement géométriques (basées sur des construction à la règle et au compas) permettent également de résoudre certains problèmes. Des techniques de décomposition structurelles ont été proposées par Jermann [62, 60, 59, 61], ces méthodes décomposent le système de contraintes en sous-systèmes vérifiant des propriétés de rigidité. Les blocks rigides sont résolus de manière indépendantes en utilisant des techniques de consistance locales. Les solutions des blocks sont alors fusionées pour obtenir les solutions globales du système.

Localisation de capteurs [38] : On considère un réseau de capteurs ayant la capacité de communiquer avec un capteur voisin situé à moins d'une distance donnée R. Parmi ces capteurs, certains ont une position fixée et d'autres sont mobiles. Le problème est de trouver une boîte englobant les positions possibles des capteurs mobiles, de manière à ce que les contraintes de proximité soient respectées. Pour résoudre ce problème, Doherty *et. al.* ont proposé un algorithme qui combine astucieusement SDP (Semi Definite Programming) et LP (Linear Programming). Chaque contrainte de distance est substituée par un inégalité de matrices linéaires conservatives de l'espace des solutions, les LMI (Linear Matrix Inequalities) :

$$\|u - v\|^2 \leq R \Longleftrightarrow \begin{bmatrix} I_2 R & (u-v) \\ (u-v)^T & R \end{bmatrix} \geq 0$$

De tels systèmes peuvent être résolus par la programmation semidéfinie ou dans certains cas par la programmation linéaire. Les bornes des domaines sont obtenues en minimisant et en maximisant chaque variable soumise au système de LMI. Toutefois, cette méthode est limitée au cas convexe, c'est-à-dire lorsque seule une borne supérieure des distances est donnée.

Circle packing [85, 115, 86] : Le problème de *Circle packing* consiste à placer n cercles de même rayon dans le carré unitaire[2]. Un *packing* est optimal lorsque le rayon des n cercle est maximal, autrement dit le problème ne peut être résolu avec un rayon supérieur. Ce problème théorique a été étudié notamment par Markòt *et. al.* [85, 86], qui ont proposé un algorithme basé sur une approximation des domaines par des polygones convexes. Leur méthode, nommée méthode des régions actives - ou MAA (Method of Active Areas), est un filtrage qui consiste à couper par des demi-plans le rectangle qui constitue initialement le domaine des points. Ces demi-plans sont calculés en fonction de la distance entre les points et également par les sommets du polygone englobant le domaine des points. Toutefois, cette méthode ne garantit pas l'exactitude des solutions et est limitée aux distances minimales. Heusch [51], généralise cet algorithme à tout type de distance (minimale, maximale et exacte), puis l'utilise pour définir une contrainte globale pour les contraintes de distance, mais sans fournir de résultats expérimentaux.

[2] le carré unitaire est défini par la boîte $[0, 1] \times [0, 1]$

Notations Dans la suite, on utilisera les notations[3] suivantes :

$\mathcal{P}_i(x_{i_1}, \ldots, x_{i_p}$	Un point de $\mathbb{R}^p$
$\mathcal{P}_i(x_i, y_i, z_i)$	Un point de $\mathbb{R}^3$
$\mathcal{P}_i(x_i, y_i)$	Un point de $\mathbb{R}^2$
$\overrightarrow{\mathcal{P}_i\mathcal{P}_j}$	le vecteur entre le point $\mathcal{P}_i$ et le point $\mathcal{P}_j$
δ_{ij}	la distance entre le point $\mathcal{P}_i$ et le point $\mathcal{P}_j$
$\boldsymbol{\mathcal{P}}_i$	le domaine du point $\mathcal{P}_i$, c'est-à-dire, $\boldsymbol{\mathcal{P}}_i = \boldsymbol{x}_{i_1} \times \ldots \times \boldsymbol{x}_{i_p}$

Pour simplifier la notation en dimension 2 on notera les contraintes de la manière suivante :

$$c_{ij} : (x_i - x_j)^2 + (y_i - y_j)^2 = \delta_{ij}^2$$

De la même manière, en dimension 3, on notera $\mathcal{P}_i(x_i, y_i, z_i)$ et les contraintes par :

$$c_{ij} : (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 = \delta_{ij}^2$$

Dans les deux chapitres suivants, nous présentons deux approches différentes pour définir une contrainte globale pour les contraintes de distance. La première est basée sur l'introduction de contraintes additionnelles inférées à partir du système initial et l'utilisation de techniques de filtrage classique. La deuxième approche est un algorithme de filtrage dédié basé sur une linéarisation du système.

[3] Un récapitulatif des notations utilisées dans ce manuscrit est présenté à la page x

Chapitre 5

Ajout de contraintes redondantes

Dans ce chapitre, nous présentons une approche originale pour définir une contrainte globale pour les systèmes de contraintes de distance en utilisant des propriétés géométriques élémentaires. Nous décrivons deux types d'informations permettant d'augmenter l'efficacité et la qualité du filtrage par consistance sur les systèmes d'équations de distance : les inégalités triangulaires et les barycentres.

Plus précisément, dans la section 5.1, nous présentons une technique permettant de calculer un domaine pour toutes les distances inconnues du système. Dans la section 5.2, nous ajoutons des points particuliers, les barycentres des triangles, dans le but d'améliorer la puissance du filtrage par consistance locale.

5.1 Fermeture du graphe de contraintes

Un problème de satisfaction de contraintes de distance est la donnée de m distances entre des couples de points. Ces distances peuvent être fixées, c'est-à-dire, δ_{ij} sont des nombre réel. Mais ces distances peuvent également être approximée par un intervalle auquel cas elles constituent des variables à part entière du CSP. En général, les distances entre tout couples de points ne sont pas connues, sinon le système serait facile à résoudre. Nous proposons dans cette section une méthode pour borner les distances inconnues du système.

Une technique de filtrage par consistance permet de réduire les domaines des points jusqu'à ce que leurs bornes vérifient la propriété de consistance associée. En général, les consistances utilisées sont locales, c'est-à-dire qu'elles mettent en jeu les contraintes indépendemment les unes des autres. La 2B-consistance va par exemple garantir que chaque contrainte de distance peut être satisfaite lorsqu'une des variables est instanciée à l'une des bornes de son domaine.

Or, en général, satisfaire chaque contrainte indépendemment ne permet pas de construire une solution, comme l'illustre l'exemple suivant :

Exemple 5.1.1 *Soient* 3 *points* $\mathcal{P}_1(x_1, y_1)$, $\mathcal{P}_2(x_2, y_2)$ *et* $\mathcal{P}_3(x_3, y_3)$, *dont les domaines respectifs sont* $\boldsymbol{\mathcal{P}}_1 = \boldsymbol{\mathcal{P}}_2 = \boldsymbol{\mathcal{P}}_3 = [-1, 1] \times [-1, 1]$. *Supposons que l'on cherche à déterminer ces points de manière à ce que les contraintes de distance suivantes soient respectées :*

$$\begin{aligned} c_{12} &: (x_1 - x_2)^2 + (y_1 - y_2)^2 = \delta_{12}^2 \\ c_{13} &: (x_1 - x_3)^2 + (y_1 - y_3)^2 = \delta_{13}^2 \\ c_{23} &: (x_2 - x_3)^2 + (y_2 - y_3)^2 = \delta_{23}^2 \end{aligned}$$

avec $\delta_{12} = 1$, $\delta_{23} = 1$ *et* $\delta_{13} = 2.1$.

Ce système est 2B-consistant car chacune des contraintes est vérifiée indépendemment (il est même 3B-consistant). Pourtant, il n' pas de solutions : *Realpaver* [46], un des solveurs de contraintes numériques les plus efficaces, requiert près de 40000 bisections pour prouver que ce problème n'a pas de solutions en utilisant HC4.

Or, pour prouver l'inconsistance de ce problème, il suffit considérer l'une des propriétés les plus élémentaires des triangles : "*La somme des longueurs de* 2 *cotés d'un triangle est supérieure ou égale à la longueur du* 3e *côté*". Cette propriété n'est pas vérifiée puisque $\delta_{12} + \delta_{23} = 2$ alors que $\delta_{13} = 2.1$. En fait, en ajoutant la contrainte $\delta_{12} + \delta_{23} \geq \delta_{13}$, un seul filtrage par 2B-consistance permet de prouver que le système n'a pas de solutions.

De manière générale, pour qu'un système de contraintes de distance ait des solutions, il est nécessaire que les distances données vérifient les inégalités triangulaires. Ces inégalités sont définies par :

$$\forall i, j, k \in [1..n] : i < j < k \begin{cases} \delta_{ij} \leq \delta_{ik} + \delta_{jk} \\ \delta_{ij} \geq |\delta_{ik} - \delta_{jk}| \end{cases} \tag{5.1}$$

Ces inégalités mettent en jeu toutes les distances du système, c'est-à-dire les m distances données et les distances inconnues. Afin de vérifier ces inégalités, il faut ajouter au système initial des variables supplémentaires pour les distances inconnues ainsi que les contraintes d'inégalités triangulaires (5.1).

Cette opération, que l'on nommera par la suite *Fermeture du graphe de contraintes*, est illustrée par la figure 5.1. A gauche de cette figure, le graphe de contraintes correspondant au problème de l'exemple 5.1.2 est représenté. Les noeuds du graphe correspondent aux points du système et les arêtes aux contraintes de distance. Une arête entre deux sommets $\mathcal{P}_i$ et $\mathcal{P}_j$ est valuée par la distance δij ou par son domaine si c'est un intervalle. Les distance inconnues sont représentées en pointillés. L'opération de fermeture permet de calculer des intervalles pour les distances manquantes. Les bornes des intervalles vérifient les inégalités triangulaires (5.1). L'instance résultant de cette fermeture, que l'on notera par la suite $\mathcal{I}_f$, possède un graphe de contraintes complet.

Exemple 5.1.2 *Considérons un système à cinq points* $ABCDE$ *:* $A(0,0)$, $B(8,0)$, $C(x_C, y_C)$, $D(x_D, y_D)$, $E(x_E, y_E)$. 5 *distances sont données :* $BC = CE = DE = 2$, $AD = 3$ *et* $CD = 4$. *Les domaines des points* C, D *et* E *sont égaux à la boîte* $[-20, 20] \times [-20, 20]$. *On a donc le système sous-contraint suivant :*

c_{AD} : $\quad {x_D}^2 + {y_D}^2 = 9$
c_{BC} : $\quad (x_C - 8)^2 + {y_C}^2 = 4$
c_{CE} : $\quad (x_C - x_E)^2 + (y_C - y_E)^2 = 4$
c_{CD} : $\quad (x_D - x_C)^2 + (y_D - y_C)^2 = 16$
c_{DE} : $\quad (x_D - x_E)^2 + (y_D - y_E)^2 = 4$

où les variables sont les coordonnées des points C, D *et* E.

Dans la section suivante, nous présentons différentes approches pour calculer la fermeture du graphe de contraintes de distance.

5.1.1 Calcul de la fermeture

Une première approche pour calculer la fermeture du graphe de contraintes consiste à ajouter les variables et les contraintes d'inégalités triangulaires (5.1) au système initial, puis

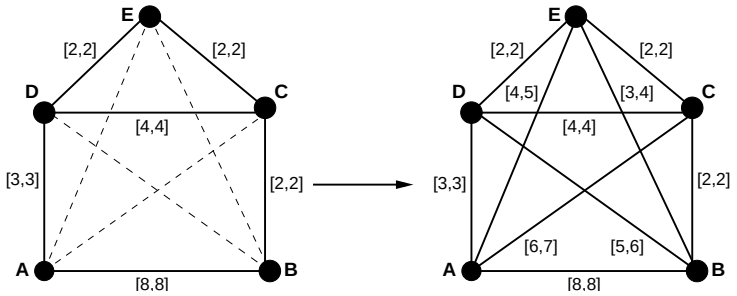


FIG. 5.1 – Fermeture du graphe de contraintes représentant le DCSP de l'exemple 5.1.2

de faire appel à la 2B-consistance pour calculer des bornes pour les distances manquantes. On obtient un filtrage plus fort des domaines comme le montre le tableau 5.1. Dans ce tableau, les deux premières colonnes montre le résultat du filtrage par 2B-consistance sur l'instance $\mathcal{I}$ de l'exemple 5.1.2 et le résultat du filtrage par 2B-consistance sur $\mathcal{I}_f$. La réduction du domaine $\boldsymbol{y}_E$ est plus forte.

	2B($\mathcal{I}$)	2B($\mathcal{I}_f$)	BSFilter2B($\mathcal{I}$)
$\boldsymbol{x}_C$	[2,3]	[2,3]	[2,3]
$\boldsymbol{y}_C$	[-2.23,2.23]	[-2.23,2.23]	[-2.23,2.23]
$\boldsymbol{x}_D$	[6,7]	[6,7]	[6,7]
$\boldsymbol{y}_D$	[-1.73,1.73]	[-1.73,1.73]	[-1.73,1.73]
$\boldsymbol{x}_E$	[4,5]	[4,5]	[4,5]
$\boldsymbol{y}_E$	[-3.46,3.46]	[-2.64,2.64]	[-2.64,2.64]
CPU time	0.12s	0.12s	0.03s

TAB. 5.1 – Comparaison des filtrages avec et sans fermeture du graphe de contraintes

Contrainte globale pour les inégalités triangulaires Une deuxième approche pour calculer la fermeture du graphe de contraintes consiste à utiliser un algorithme spécifique pour résoudre les inégalités triangulaires. Cet algorithme, nommé BoundSmoothing, a été introduit par Crippen et Havel [31].

Pour chaque arête du graphe, l'algorithme parcourt tous les triangles du graphe contenant cette arête, en mettant à jour les bornes du domaine de sa longueur. C'est une modification de l'algorithme du plus court chemin pour tout couple de sommets de Floyd qui a une complexité en temps de $\Theta(n^3)$. Une preuve et une analyse de cet algorithme se trouvent dans le recueil d'algorithmes de Cormen, Leiserson et Rivest [30].

Nous avons introduit dans l'algorithme BoundSmoothing une procédure de filtrage qui réduit les domaines des points à la volée. Ce algorithme, nommé BSFilter (voir figure 5.2), a la même structure que l'algorithme BoundSmoothing. Chaque fois qu'un triangle $\mathcal{P}_i\mathcal{P}_j\mathcal{P}_k$ est examiné, les domaines des sommets du triangles sont filtrés par la procédure FilterTriangle (ligne 15). Deux implantation de FilterTriangle ont été expérimentées donnant lieu à deux variantes de BSFilter : BSFilter2B et BSFilter3B. Ces deux procédures correspondent respectivement à un filtrage par 2B-consistance et par 3B-consistance du sous-système constitué par le triangle $\mathcal{P}_i\mathcal{P}_j\mathcal{P}_k$.

```
BSFilter(in-out : (𝒳, 𝒟, 𝒞))
1 :  foreach k ≤ n do
2 :    foreach j ≤ n do
3 :      foreach i ≤ n do
4 :        if δ̄_ij > δ̄_ik + δ̄_jk then
4 :        if δ̄_ij > δ̄_ik + δ̄_jk then
5 :          δ̄_ij = δ̄_ik + δ̄_jk
6 :        endif
7 :        if δ_ij < δ_ik − δ̄_jk then
8 :          δ_ij = δ_ik − δ̄_jk
9 :        else if δ_ij < δ_jk − δ̄_ik then
10 :         δ_ij = δ_jk − δ̄_ik
11 :       endif
12 :       if δ_ij > δ̄_ij then
13 :         return false
14 :       endif
15 :     FilterTriangle((𝒳, 𝒟, 𝒞), i, j, k)
16 :     endfor
17 :   endfor
18 : endfor
19 : return true
```

FIG. 5.2 – Algorithme BSFilter qui calcule la fermeture du graphe de contraintes

BSFilter permet en général d'obtenir un meilleur filtrage qu'une consistance locale sur le problème initial car il utilise plus d'informations. Toutefois, à l'instar des consistances locales, les réductions opérées par le filtrage d'un triangle ne sont pas propagées. Autrement dit, BSFilter ne permet pas d'assurer une propriété de consistance sur les bornes des domaines. Pour obtenir cette propriété, il est donc nécessaire d'appliquer successivement un pré-filtrage par BSFilter puis un filtrage par consistance. Les contraintes de distance inférée par la procédure de fermeture améliore la propagation des réductions de domaines et la puissance du filtrage par consistance locale. La section suivante présente quelques résultats expérimentaux.

5.1.2 Résultats expérimentaux

Nous avons évalué l'apport de ces contraintes redondantes sur deux problèmes : le problème des ponts et les problème des losanges [59]. Le problème des ponts est constitué de 15 points en dimension 2 et de 26 équations de distance (voir figure 5.3). Deux de ces points sont fixés, ainsi le système est régulier (autant d'inconnues que d'équations) et possède 128 solutions réelles. Les domaines des 13 autres points sont fixés à $[-100, 100] \times [-100, 100]$ et contiennent l'ensemble des solutions. Le problème des losanges est constitué de 13 points en dimension 2 et 22 équations de distance. De la même manière, 2 points sont fixés et les domaines des autres points sont $[-100, 100] \times [-100, 100]$. Ce problème a 64 solutions réelles.

Les tableau 5.2 et 5.3 montrent la réduction des domaines opérés par les différents filtrage ainsi que les temps de calcul :

- 2B($\mathcal{I}$)/3B($\mathcal{I}$) : Filtrage par 2B-consistance/3B-consistance sur l'instance initiale.
- BSFilter2B($\mathcal{I}$)/ BSFilter3B($\mathcal{I}$) : Fermeture du graphe de distance par *BSFilter* avec filtrage des triangles par 2B-consistance/3B-consistance.
- 3B($\mathcal{I}_f$) : Filtrage par 3B-consistance de l'instance fermée, c'est-à-dire l'instance initiale à laquelle on a ajouté les contraintes d'inégalités triangulaires, les contraintes de distance complémentaires et les variables supplémentaires associées.

Tous les tests ont été réalisés en utilisant ILOG Solver [54] sur un PC muni d'un processeur Pentium III cadencé à 800Mhz sous la plateforme Linux. Ces tableaux montrent un gain significatif en temps de calcul et une amélioration du filtrage. Par exemple, sur le problème des ponts (voir table 5.2), le temps de calcul pour la 3B sur le problème initial est divisé par 3 en utilisant un pré-filtrage par BSFilter3B. Sur ce problème, un filtrage plus fort a pu être réalisé notamment sur les variables y_{10} et y_{12}. On observe aussi un gain de l'ordre d'un facteur 2 en temps de calcul pour le filtrage par la 3B sur le problème fermé. Ces résultats montre que la propagation est plus efficace avec les contraintes redondantes, bien qu'un grand nombre de variables et de contraintes aient été ajoutées au problème initial (79 distances inférées, 79 contraintes de distance correspondantes et 910 inégalités triangulaires). On obtient des résultats similaires pour le problème des losanges (voir table 5.3).

5.2 Introduction de barycentres

Dans la section précédente, nous avons montré l'apport de l'inférence de contraintes de distance sur le filtrage par consistance. L'algorithme BSFilter (voir figure 5.2), calcule des intervalles de définition pour les distances manquantes qui vérifient les inégalités triangulaires. Cet algorithme parcourt tous les triangles du système pour mettre à jour les intervalles des distances calculées. Chaque fois qu'un triangle est traité, une procédure de filtrage est appelée pour réduire les domaines des sommets du triangle considéré.

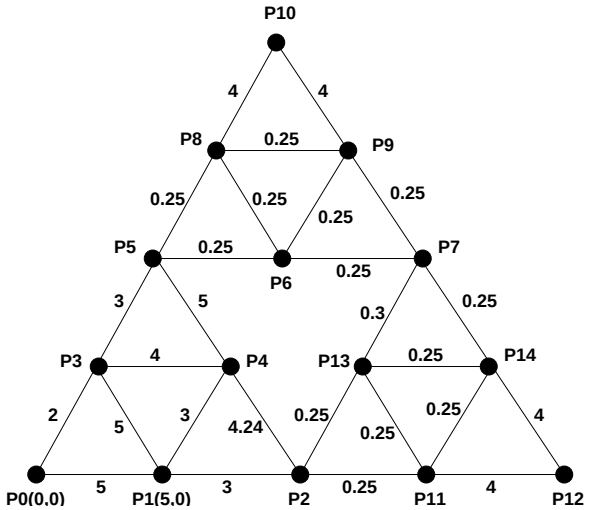


FIG. 5.3 – Problème des ponts[59]

	2B($\mathcal{I}$)	BSFilter2B($\mathcal{I}$)	3B($\mathcal{I}$)	3B($\mathcal{I}_f$)	BSFilter3B($\mathcal{I}$)	3B(BSFilter3B($\mathcal{I}$))
$\boldsymbol{x}_0$	[0.0]	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\boldsymbol{y}_0$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\boldsymbol{x}_1$	$[5,5]$	$[5,5]$	$[5,5]$	$[5,5]$	$[5,5]$	$[5,5]$
$\boldsymbol{y}_1$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\boldsymbol{x}_2$	$[2,4.46]$	$[2,4.46]$	$[2.08,3.49]$	$[2.08,3.49]$	$[2.08,3.49]$	$[2.08,3.49]$
$\boldsymbol{y}_2$	$[-2.95,2.95]$	$[-2.95,2.95]$	$[-2.59,2.59]$	$[-2.59,2.59]$	$[-2.59,2.59]$	$[-2.59,2.59]$
$\boldsymbol{x}_3$	$[0,0.41]$	$[0,0.41]$	$[0.4,0.4]$	$[0.4,0.4]$	$[0.4,0.4]$	$[0.4,0.4]$
$\boldsymbol{y}_3$	$[-2,2]$	$[-2,2]$	$[-1.96,1.96]$	$[-1.96,1.96]$	$[-1.96,1.96]$	$[-1.96,1.96]$
$\boldsymbol{x}_4$	$[2,4.41]$	$[2,4.41]$	$[2.4,4.28]$	$[2.4,4.28]$	$[2.4,4.28]$	$[2.4,4.28]$
$\boldsymbol{y}_4$	$[-2.94,2.94]$	$[-2.94,2.94]$	$[-2.91,2.91]$	$[-2.91,2.91]$	$[-2.91,2.91]$	$[-2.91,2.91]$
$\boldsymbol{x}_5$	$[0.94,3.41]$	$[0.94,3.41]$	$[1.06,3.39]$	$[1.06,3.39]$	$[1.03,3.39]$	$[1.03,3.39]$
$\boldsymbol{y}_5$	$[-4,4]$	$[-3.72,3.72]$	$[-3.46,3.46]$	$[-3.46,3.46]$	$[-3.46,3.46]$	$[-3.46,3.46]$
$\boldsymbol{x}_6$	$[1.2,3.66]$	$[1.2,3.66]$	$[1.28,3.64]$	$[1.28,3.64]$	$[1.28,3.61]$	$[1.28,3.61]$
$\boldsymbol{y}_6$	$[-3.75,3.75]$	$[-3.72,3.72]$	$[-3.30,3.30]$	$[-3.30,3.30]$	$[-3.39,3.39]$	$[-3.30,3.30]$
$\boldsymbol{x}_7$	$[1.54,3.91]$	$[1.54,3.89]$	$[1.53,3.89]$	$[1.53,3.89]$	$[1.53,3.79]$	$[1.53,3.79]$
$\boldsymbol{y}_7$	$[-3.5,3.5]$	$[-3.38,3.38]$	$[-3.13,3.13]$	$[-3.13,3.13]$	$[-3.14,3.14]$	$[-3.13,3.13]$
$\boldsymbol{x}_8$	$[1.04,3.66]$	$[1.04,3.66]$	$[1.04,3.64]$	$[1.04,3.64]$	$[1.07,3.61]$	$[1.04,3.61]$
$\boldsymbol{y}_8$	$[-4,4]$	$[-3.8,3.8]$	$[-3.54,3.54]$	$[-3.54,3.54]$	$[-3.61,3.61]$	$[-3.54,3.54]$
$\boldsymbol{x}_9$	$[1.2,3.91]$	$[1.2,3.91]$	$[1.28,3.89]$	$[1.28,3.89]$	$[1.28,3.83]$	$[1.28,3.83]$
$\boldsymbol{y}_9$	$[-3.75,3.75]$	$[-3.63,3.63]$	$[-3.37,3.37]$	$[-3.37,3.37]$	$[-3.39,3.39]$	$[-3.37,3.37]$
$\boldsymbol{x}_{10}$	$[-2.8,7.66]$	$[-2.8,7.66]$	$[-2.71,7.64]$	$[-2.71,7.64]$	$[-2.71,7.61]$	$[-2.71,7.61]$
$\boldsymbol{y}_{10}$	$[-7.75,7.75]$	$[-7.63,7.63]$	$[-7.33,7.33]$	$[\mathbf{-7.25},\mathbf{7.25}]$	$[-7.39,7.39]$	$[\mathbf{-7.25},\mathbf{7.25}]$
$\boldsymbol{x}_{11}$	$[1.75,4.41]$	$[1.75,4.41]$	$[1.83,3.74]$	$[1.83,3.74]$	$[1.83,3.74]$	$[1.83,3.74]$
$\boldsymbol{y}_{11}$	$[-3.2,3.2]$	$[-3.2,3.2]$	$[-2.84,2.84]$	$[-2.84,2.84]$	$[-2.84,2.84]$	$[-2.84,2.84]$
$\boldsymbol{x}_{12}$	$[-2.25,8.16]$	$[-2.25,8.16]$	$[-2.16,7.74]$	$[-2.16,7.74]$	$[-2.16,7.74]$	$[-2.16,7.74]$
$\boldsymbol{y}_{12}$	$[-7.2,7.2]$	$[-7.19,7.19]$	$[-6.83,6.83]$	$[\mathbf{-6.69},\mathbf{6.69}]$	$[-6.84,6.84]$	$[\mathbf{-6.69},\mathbf{6.69}]$
$\boldsymbol{x}_{13}$	$[1.75,4.21]$	$[1.75,4.21]$	$[1.83,3.74]$	$[1.83,3.74]$	$[1.83,3.74]$	$[1.83,3.74]$
$\boldsymbol{y}_{13}$	$[-3.2,3.2]$	$[-3.15,3.15]$	$[-2.84,2.84]$	$[-2.84,2.84]$	$[-2.84,2.84]$	$[-2.84,2.84]$
$\boldsymbol{x}_{14}$	$[1.5,4.16]$	$[1.5,4.16]$	$[1.58,3.99]$	$[1.58,3.99]$	$[1.62,3.94]$	$[1.58,3.94]$
$\boldsymbol{y}_{14}$	$[-3.45,3.45]$	$[-3.39,3.39]$	$[-3.03,3.03]$	$[-3.03,3.03]$	$[-3.06,3.06]$	$[-3.03,3.03]$
CPU	$0.30s$	$0.17s$	1141.37	**673.69s**	**284.78s**	**464.78s**

TAB. 5.2 – Résultats des différents filtrages pour le problème des ponts

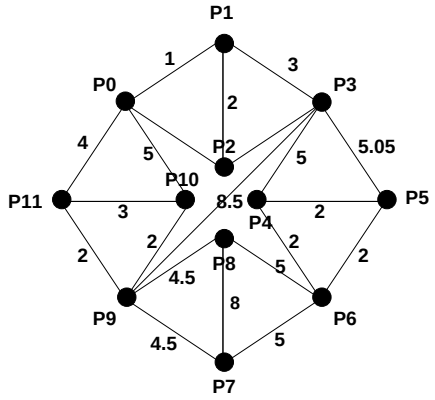


FIG. 5.4 – Le problème des losanges [59]

	2B($\mathcal{I}$)	BSFilter2B($\mathcal{I}$)	3B($\mathcal{I}$)	BSFilter3B($\mathcal{I}$)	3B(BSFilter3B($\mathcal{I}$))
$\boldsymbol{x}_0$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\boldsymbol{y}_0$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\boldsymbol{x}_1$	$[1,1]$	$[1,1]$	$[1,1]$	$[1,1]$	$[1,1]$
$\boldsymbol{y}_1$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$	$[0,0]$
$\boldsymbol{x}_2$	$[-1,2]$	$[-1,2]$	$[0.49,0.50]$	$[0.5,0.5]$	$[0.49,0.50]$
$\boldsymbol{y}_2$	$[-2,2]$	$[-2,2]$	$[-1.93,1.93]$	$[-1.93,1.93]$	$[-1.93,1.93]$
$\boldsymbol{x}_3$	$[-2,4]$	$[-2,4]$	$[3.48,3.48]$	$[3.48,3.48]$	$[3.48,3.48]$
$\boldsymbol{y}_3$	$[-3,3]$	$[-3,3]$	$[-1.67,1.67]$	$[-1.67,1.67]$	$[-1.67,1.67]$
$\boldsymbol{x}_4$	$[-7,9]$	$[-7,9]$	$[-1.51,8.48]$	$[-1.51,8.48]$	$[-1.51,8.48]$
$\boldsymbol{y}_4$	$[-8,8]$	$[-8,8]$	$[-6.67,6.67]$	$[-6.67,6.67]$	$[-6.67,6.67]$
$\boldsymbol{x}_5$	$[-7.05,9.05]$	$[-7.05,9.05]$	$[-1.56,8.53]$	$[-1.56,8.53]$	$[-1.56,8.53]$
$\boldsymbol{y}_5$	$[-8.05,8.05]$	$[-8.05,8.05]$	$[-6.72,6.72]$	$[-6.72,6.72]$	$[-6.72,6.72]$
$\boldsymbol{x}_6$	$[-9,11]$	$[-9,11]$	$[-3.51,8.66]$	$[-3.26,7.33]$	$[\mathbf{-3.26},\mathbf{7.33}]$
$\boldsymbol{y}_6$	$[-10,10]$	$[-10,10]$	$[-7.47,7.47]$	$[-8.43,8.43]$	$[-7.47,7.47]$
$\boldsymbol{x}_7$	$[-10.5,10.5]$	$[-10.5,10.5]$	$[-8.51,3.66]$	$[-8.26,3.98]$	$[\mathbf{-8.26},\mathbf{3.66}]$
$\boldsymbol{y}_7$	$[-10.5,10.5]$	$[-10.5,10.5]$	$[-9.62,9.62]$	$[-10.32,10.32]$	$[-9.62,9.62]$
$\boldsymbol{x}_8$	$[-10.5,10.5]$	$[-10.5,10.5]$	$[-8.51,3.66]$	$[-8.26,3.98]$	$[\mathbf{-8.26},\mathbf{3.66}]$
$\boldsymbol{y}_8$	$[-10.5,10.5]$	$[-10.5,10.5]$	$[-9.62,9.62]$	$[-10.32,10.32]$	$[-9.62,9.62]$
$\boldsymbol{x}_9$	$[-6,6]$	$[-6,6]$	$[-5.01,-0.84]$	$[-5.01,-0.84]$	$[-5.01,-0.84]$
$\boldsymbol{y}_9$	$[-6,6]$	$[-6,6]$	$[-5.64,5.64]$	$[-5.82,5.82]$	$[-5.64,5.64]$
$\boldsymbol{x}_{10}$	$[-5,5]$	$[-5,5]$	$[-5.00,1.01]$	$[-5,1.30]$	$[-5.00,1.01]$
$\boldsymbol{y}_{10}$	$[-5,5]$	$[-5,5]$	$[-4.99,4.99]$	$[-5,5]$	$[-4.99,4.99]$
$\boldsymbol{x}_{11}$	$[-4,4]$	$[-4,4]$	$[-4.00,0.29]$	$[-4,0.30]$	$[-4.00,0.29]$
$\boldsymbol{y}_{11}$	$[-4,4]$	$[-4,4]$	$[-3.99,3.99]$	$[-4,4]$	$[-3.99,3.99]$
CPU	$0.23s$	$0.11s$	$649.24s$	**116s**	**266s**

TAB. 5.3 – Résultats des différents filtrages pour le problème des losanges

Nous avons présenté deux implantations différentes de la procédure FilterTriangle qui appliquent un filtrage par consistance du sous-système constitué des sommets du triangle (BSFilter2B et BSFilter3B). Nous présentons dans cette section, une nouvelle variante de la procédure FilterTriangle. Cette variante utilise des propriétés élémentaires sur les triangles pour inférer de nouvelles contraintes de distance. Ces contraintes supplémentaires permettent un filtrage plus fort des domaines des points.

- $\mathcal{X} = \{x_i, y_i, x_j, y_j, x_k, y_k\}$
- $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{x}_j, \boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{y}_k\}$
- $\mathcal{C} = \begin{cases} c_{ij} : (x_i - x_j)^2 + (y_i - y_j)^2 = \delta_{ij}^2 \\ c_{ik} : (x_i - x_k)^2 + (y_i - y_k)^2 = \delta_{ik}^2 \\ c_{jk} : (x_j - x_k)^2 + (y_j - y_k)^2 = \delta_{jk}^2 \end{cases}$

FIG. 5.5 – Description du CSP T_{ijk}

Plus précisément, on considère un système à 3 contraintes de distance et 3 points ($\mathcal{P}_i$,$\mathcal{P}_j$ et $\mathcal{P}_k$), avec l'hypothèse que les domaines des distances satisfont les inégalités triangulaires. L'idée est d'ajouter à ce système un point supplémentaire, le centre de gravité - ou *isobarycentre* du triangle - que l'on notera $G(x_G, y_G)$. Le domaine de ce point est facile à calculer en utilisant sa définition ($3x_G = x_i + x_j + x_k$, $3y_G = y_i + y_j + y_k$). D'autre part, les distances entre G et chacun des sommets du triangle sont liées par des équations linéaires (voir propriété 5.2.1). La relation entre les longueurs des arêtes du triangles et les distances entre G et les sommets sont définis sous les hypothèses que les longueurs des arêtes vérifient les inégalités triangulaires. Plus formellement,

Proposition 5.2.1 *Soient trois points A, B et C et soit G le centre de gravité du triangle ABC. Alors, si u, v et w sont strictement positifs on a :*

$$\begin{array}{lcl} GA^2 = \frac{1}{9}(2u^2 + 2v^2 - w^2) & & AB = u \\ GB^2 = \frac{1}{9}(2u^2 + 2w^2 - v^2) & \Longleftrightarrow & AC = v \\ GC^2 = \frac{1}{9}(2v^2 + 2w^2 - u^2) & & BC = w \end{array}$$

Une démonstration de cette propriété se trouve dans l'annexe A page 107. Cette propriété permet de définir le CSP T'_{ijk}(c.f. Fig 5.6). Les solutions de T_{ijk} satisfont les contraintes de T'_{ijk} et inversement. Le point clé est que les distance entre le centre de gravité et les sommets du triangle encapsulent des informations sur les angles entre les segments du triangle (plus précisément sur les produits scalaires). Un filtrage par consistance sur le CSP T'_{ijk} est donc beaucoup plus informé qu'un filtrage sur T_{ijk}.

Le tableau 5.4 montre les résultats obtenus pour le problème de l'exemple 5.1.2 en filtrant les T'_{ijk} avec une 3B-consistance. Dans cet exemple, on avait $DE = CE = 2$ et $CD = 4$, ce qui signifie que E est le milieu du segment CD. Rien ne permet a priori aux techniques de consistances locales de détecter cette propriété particulière du point E. Par contre, si l'on introduit le barycentre G du triangle CDE, on calcule les distances GC, GD et GE en utilisant les formules de la propriétés 5.2.1 et on obtient : $GC = 2$, $GD = 2$ et $GE = 0$. Cela indique clairement que E est confondu avec G, et un filtrage par consistance du triangle CDE avec ces contraintes supplémentaires permet de traduire cette propriété sur les domaines

- $\mathcal{X} = \{x_i, y_i, x_j, y_j, x_k, y_k, x_G, y_G\}$
- $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{x}_j, \boldsymbol{y}_j, \boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{x}_G, \boldsymbol{y}_G\}$ tels que
  - $\boldsymbol{x}_G = \frac{1}{3}(\boldsymbol{x}_i + \boldsymbol{x}_j + \boldsymbol{x}_k)$
  - $\boldsymbol{y}_G = \frac{1}{3}(\boldsymbol{y}_i + \boldsymbol{y}_j + \boldsymbol{y}_k)$.
- $\mathcal{C} = \left\{ \begin{array}{rcl} (x_G - x_i)^2 + (y_G - y_i)^2 & = & \frac{1}{9}(2\delta_{ik}^2 + 2\delta_{ij}^2 - \delta_{jk}^2) \\ (x_G - x_j)^2 + (y_G - y_j)^2 & = & \frac{1}{9}(2\delta_{ij}^2 + 2\delta_{jk}^2 - \delta_{ik}^2) \\ (x_G - x_k)^2 + (y_G - y_k)^2 & = & \frac{1}{9}(2\delta_{ik}^2 + 2\delta_{jk}^2 - \delta_{ij}^2) \\ x_G & = & \frac{1}{3}(x_i + x_j + x_k) \\ y_G & = & \frac{1}{3}(y_i + y_j + y_k) \end{array} \right.$

FIG. 5.6 – Description du CSP T'_{ijk}

des points. La figure 5.7 montrent qu'en effet le domaine du point E est égal à la demi-somme des domaines du point C et du point D. Les résultats numériques obtenus confirment la propriété attendue, comme le montre la table 5.4, on a bien : $\boldsymbol{x}_E = [-1.76, 1.76]$ et $(\boldsymbol{x}_C + \boldsymbol{x}_D)/2 = ([-1.61, 1.61] + [-1.91, 1.91])/2 = [-1.76, 1.76]$. Ce filtrage est donc meilleur que celui de la 3B, puisqu'il tire parti de propriétés géométriques spécifique au système de contraintes.

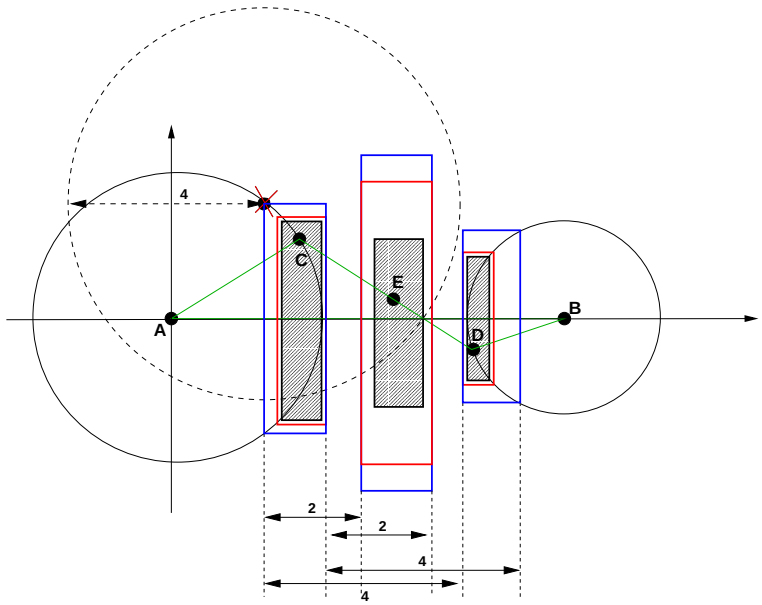


FIG. 5.7 – Comparaison des résultats pour la 2B-consistance, la 3B-consistance et la 3B-consistance sur tous les triangles avec introduction du barycentre.

	3B($\mathcal{I}$)	3B($\mathcal{I}_f$)	BSFilter3BBary($\mathcal{I}_f$
D_{x_C}	$[2.31, 3]$	$[2.31, 3]$	$[2.31, 3]$
D_{y_C}	$[-1.91, 1.91]$	$[-1.91, 1.91]$	$[-1.91, 1.91]$
D_{x_D}	$[6, 6.81]$	$[6, 6.81]$	$[6, 6.81]$
D_{y_D}	$[-1.61, 1.61]$	$[-1.61, 1.61]$	$[-1.61, 1.61]$
D_{x_E}	$[4, 5]$	$[4, 5]$	$[4.15, 4.9]$
D_{y_E}	$[-2.14, 2.14]$	$[-1.82, 1.82]$	$[-1.76, 1.76]$

Tab. 5.4 – Réduction des domaines du CSP de l'exemple 5.1.2

Chapitre 6

QuadDist : Un filtrage global pour les contraintes de distance

Ce chapitre présente un algorithme de filtrage global pour résoudre les systèmes d'équations de distance. Cette nouvelle technique nommée QuadDist est dérivée de Quad, une technique de filtrage global pour les contraintes quadratiques. Quad calcule une relaxation linéaire des termes quadratiques des équations et utilise l'algorithme du simplexe pour réduire les domaines des variables. Nous présentons dans ce chapitre une approximation linéaire dédiée aux équations de distance euclidienne. Le point clé de cette nouvelle méthode réside dans le fait que ces approximations ne sont pas générées pour chaque terme des équations mais globalement pour chaque contrainte de distance. QuadDist définit donc une approximation plus précise que Quad, ce qui augmente la vitesse de convergence de la méthode. D'autre part, contrairement à la Quad, notre linéarisation des contraintes ne nécessite pas l'ajout de variables supplémentaires ce qui réduit la taille des systèmes linéaires résolus par le simplexe. Les résultats expérimentaux montrent que QuadDist résout certains problèmes près de 300 fois plus vite que la Quad.

6.1 Introduction

Les outils classiques pour résoudre les contraintes numériques sont basées sur des consistances locales comme la 2B-consistance [79, 80, 13] ou la Box-consistance [14, 119]. L'inconvénient de ces méthodes vient du fait que les contraintes sont traitées indépendamment et sans exploiter leurs propriétés sémantiques.

Lebbah *et al* ont introduit un algorithme de filtrage global, nommé Quad, pour traiter les contraintes quadratiques [76]. Quad génère une relaxation linéaire des termes quadratiques des équations puis d'utiliser l'algorithme du simplexe pour réduire les domaines des variables.

Nous présentons dans ce chapitre une approximation linéaire dédiée aux équations de distance euclidienne. Le point clé de cette nouvelle méthode réside dans le fait que ces approximations ne sont pas générées pour chaque terme des équations mais globalement pour chaque contrainte de distance. QuadDist définit donc une approximation plus précise que Quad. Contrairement à la Quad notre linéarisation des contraintes ne nécessite pas l'ajout de variables additionnelles, ce qui réduit la taille des systèmes linéaires engendrés. Les performances de QuadDist sur des CSP constitués d'équations de distance sont tout à fait encourageantes.

Heusch [51] a introduit une contrainte globale pour traiter les relations de distance entre n points mais sans toutefois fournir de résultats expérimentaux. Son approche est basée sur la méthode des "Active Areas" utilisées par Markót et Csendes [85]. L'avantage de QuadDist est qu'il est s'agit d'un schéma plus général et plus facile à mettre en œuvre.

La section suivante présente les principes de QuadDist sur un exemple très simple.

6.1.1 Présentation informelle

Considérons le système de contraintes suivant :

$$\mathcal{C} = \begin{cases} c_1 : & x^2 + y^2 = 10 \\ c_2 : & (x-4)^2 + y^2 = 10 \end{cases}$$

et supposons que les domaines des variables x et y soient $\boldsymbol{x} = [1,3]$ et $\boldsymbol{y} = [1,4]$. La solution de $\mathcal{C}$ (voir figure 6.1) est le point d'intersection de deux cercles de rayon $\sqrt{10}$: $\mathcal{C}_1$ centré en $(0,0)$ et $\mathcal{C}_2$ centré en $(4,0)$.

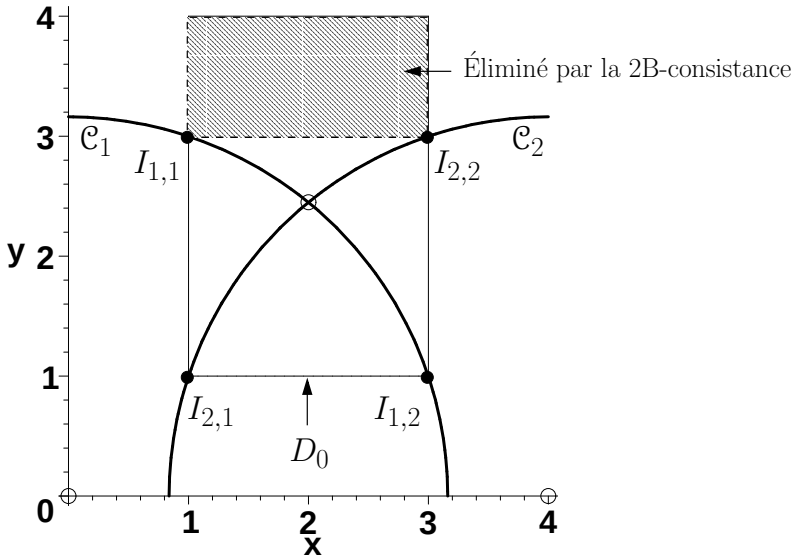


FIG. 6.1 – Réduction du domaine par 2B-consistance

La 2B-consistance réduit le domaine initial $D_0 = \boldsymbol{x} \times \boldsymbol{y}$ à $[1,3] \times [1,3]$ puisque les bornes des domaines des variables satisfont les contraintes c_1 et c_2. La figure 6.1 montre en effet que les points $I_{1,1}$ et $I_{1,2}$ satisfont $c1$ et les points $I_{2,1}$ et $I_{2,2}$ satisfont $c2$. Pour réduire davantage les domaines, il faut utiliser une consistance plus forte ou une méthode d'énumération comme la bissection.

Le processus de relaxation de QuadDist (détaillé dans la section 3) calcule l'ensemble $\mathcal{L}(\mathcal{C})$ des approximations linéaires des contraintes de c_1 et c_2 :

$$\mathcal{L}(\mathcal{C}) = \underline{\mathcal{L}}(c_1) \cup \underline{\mathcal{L}}(c_2) \cup \overline{\mathcal{L}}(c_1) \cup \overline{\mathcal{L}}(c_2)$$

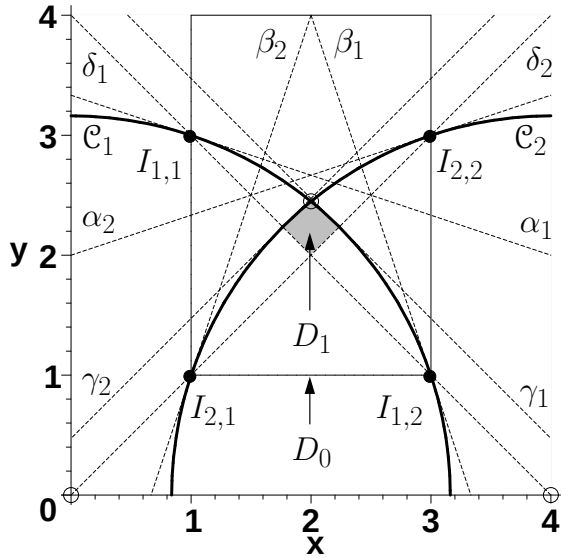


FIG. 6.2 – Première approximation des solutions de $\mathcal{L}(\mathcal{C})$.

où $\underline{\mathcal{L}}(c_i)$ est l'ensemble des sous-estimateurs de la contrainte c_i et $\overline{\mathcal{L}}(c_i)$ est l'ensemble des surestimateurs de c_i. Plus précisément :

- $\overline{\mathcal{L}}(c_i)$ est défini par trois tangentes au cercle $\mathcal{C}_i$:
  - α_i et β_i, les tangentes à $\mathcal{C}_i$ aux points $I_{i,1}$ et $I_{i,2}$, les points d'intersection entre $\mathcal{C}_i$ et la boîte définie par les domaines initiaux de x et y.
  - γ_i, la tangente au milieu de l'arc de cercle $\widehat{I_{i,1}I_{i,2}}$.
- $\underline{\mathcal{L}}(c_i)$ est la corde δ_i entre les points $I_{i,1}$ et $I_{i,2}$.

Plus formellement, le système linéaire $\mathcal{L}(\mathcal{C})$ généré par QuadDist est le suivant :

$$\mathcal{L}(\mathcal{C}) = \begin{cases} \alpha_1 & : & 3y + x - 10 \leq 0 \\ \beta_1 & : & y + 3x - 10 \leq 0 \\ \gamma_1 & : & 2y + 2x - \sqrt{80} \leq 0 \\ \delta_1 & : & y + 2x - 8 \geq 0 \\ & & \\ \alpha_2 & : & 3y - (x-4) - 10 \leq 0 \\ \beta_2 & : & y - 3(x-4) - 10 \leq 0 \\ \delta_2 & : & 2y - 2(x-4) - 8 \geq 0 \\ \gamma_2 & : & 2y - 2(x-4) - \sqrt{80} \leq 0 \end{cases}$$

Le simplexe est utilisé pour calculer les valeurs minimales et maximales de x et y satisfaisant les contraintes linéaires du système $\mathcal{L}(\mathcal{C})$. Après cette étape de filtrage, le domaine initial D_0 est réduit à la boîte englobant D_1 (voir figure6.2), puis le processus de relaxation recommence. Quatre itérations supplémentaires permettent d'isoler la solution du système avec une précision de 6 chiffres décimaux.

Dans la suite de ce chapitre, nous rappellerons les grandes lignes de l'algorithme Quad. La section 3 détaille le principe général de notre approche ainsi que le processus de linéarisation en

dimension 2. Dans la section 4 la taille des programmes linéaires générés par QuadDist est comparé à ceux que génère la Quad. Enfin, la section 4 présente quelques résultats expérimentaux.

6.2 Quad :un filtrage global pour les contraintes quadratiques

Quad est un algorithme de filtrage global pour les systèmes de contraintes quadratiques (*c.f* [76, 89, 75] pour une description plus détaillée). Cet algorithme repose sur une approximation des contraintes quadratiques du système initial par un ensemble de contraintes linéaires. Le simplexe est utilisé pour calculer une borne inférieure et supérieure des variables.

La relaxation des contraintes quadratiques consiste d'abord à introduire une nouvelle variable pour chaque terme non linéaire des contraintes quadratiques. Les domaines des variables additionnelles sont calculés en utilisant l'arithmétique de base sur les intervalles. Les termes quadratiques sont ensuite approximés par deux classes de relaxations linéaires. Ces relaxations génèrent des surestimateurs et des sous-estimateurs linéaires qui définissent une enveloppe convexe des termes quadratiques. Ces approximations apportent une meilleure estimation de l'ensemble des solutions que les fonctions de projection basées sur l'arithmétique des intervalles.

Nous montrons dans ce chapitre que ces relaxations peuvent être améliorées de manière significative en construisant une approximation linéaire globalement pour chaque contrainte de distance plutôt qu'en combinant les approximations de ses termes quadratiques.

Dans la section suivante, nous montrons comment le processus de relaxation de QuadDist exploite la sémantique des contraintes de distance pour calculer une approximation plus précise que la Quad sans introduire de variables additionnelles.

6.3 QuadDist : Une approche globale

QuadDist est une contrainte globale qui a été conçue pour résoudre efficacement les problèmes mettant en jeu des contraintes de distance dans le plan et dans l'espace tridimensionnel. Pour simplifier la description de notre méthode, nous détaillerons notre approche en dimension 2 où les illustrations sont plus intuitives.

6.3.1 Schéma général

QuadDist calcule une réduction globale des domaines en itérant les deux étapes suivantes jusqu'à atteindre un point fixe :

1. **Relaxation des contraintes** : Génération de $\mathcal{L}(\mathcal{C})$ l'ensemble des approximations linéaires des contrainte $c_{ij} \in \mathcal{C}$ de la forme :

$$c_{ij} : (x_j - x_i)^2 + (y_j - y_i)^2 = \delta_{ij}^2$$

où x_i et y_i (respectivement x_j et y_j) sont les coordonnées du point $\mathcal{P}_i$ (respectivement du point $\mathcal{P}_j$). Ces approximations linéaires sont de la forme :

$$a(x_j - x_i) + b(y_j - y_i) + c \leq 0$$

où a, b et c sont des nombres réels.

2. **Filtrage des domaines** : Utilisation du simplexe pour minimiser et maximiser chaque x_i et y_i soumis aux contraintes linéaires du système $\mathcal{L}(\mathcal{C})$.

Ce processus itératif s'arrête lorsque toutes les réductions de domaines sont plus petites qu'un certain réel positif ϵ.

Pour calculer les relaxations linéaires, les contrainte c_{ij} sont mises sous forme canonique en effectuant la substitution suivante :

$$x = x_j - x_i$$
$$y = y_j - y_i$$

où x et y sont des variables temporaires dont les domaines $\boldsymbol{x}$ et $\boldsymbol{y}$ sont calculés en utilisant l'arithmétique de base des intervalles. La contrainte c_{ij} s'écrit donc :

$$c_{ij} : x^2 + y^2 = \delta_{ij}^2$$

Les domaines de x et de y et la distance δ_{ij} sont utilisés pour déterminer les coefficients a, b et c des relaxations linéaires de c_{ij} ; ces relaxations sont de la forme :

$$ax + by + c \leq 0$$

où a, b et c sont des nombres réels. Enfin, on génère les relaxations de la contrainte c_{ij} qui sont effectivement ajoutées au système de contraintes ; à savoir :

$$a(x_j - x_i) + b(y_j - y_i) + c \leq 0$$

Le problème se ramène donc à calculer l'ensemble $\mathcal{L}(c)$ des approximations linéaires d'une contrainte sous forme canonique $c : x^2 + y^2 = \delta^2$. Ce problème est loin d'être trivial car l'ensemble des solutions de c n'est en général ni convexe ni monotone. La section suivante montre comment décomposer les domaines de x et y de manière à ce que ces propriétés soient vérifiées, tandis que la section 6.3.3 montre comment calculer les relaxations de c.

6.3.2 Décomposition sémantique des domaines

Considérons la contrainte $c : x^2 + y^2 = \delta^2$ avec $\mathcal{P}(x, y) \in \boldsymbol{\mathcal{P}} = \boldsymbol{x} \times \boldsymbol{y}$ et $\delta > 0$. $\boldsymbol{\mathcal{P}}$ est décomposé en au plus quatre sous-domaines sur lesquels la contrainte c est monotone et convexe :

$$\boldsymbol{\mathcal{P}}^{++} = \square_{\mathbb{I}^2}\{(x, y) \in \boldsymbol{\mathcal{P}} / x \geq 0, y \geq 0, x^2 + y^2 = \delta^2\}$$

$$\boldsymbol{\mathcal{P}}^{-+} = \square_{\mathbb{I}^2}\{(x, y) \in \boldsymbol{\mathcal{P}} / x \geq 0, y \leq 0, x^2 + y^2 = \delta^2\}$$

$$\boldsymbol{\mathcal{P}}^{+-} = \square_{\mathbb{I}^2}\{(x, y) \in \boldsymbol{\mathcal{P}} / x \leq 0, y \geq 0, x^2 + y^2 = \delta^2\}$$

$$\boldsymbol{\mathcal{P}}^{--} = \square_{\mathbb{I}^2}\{(x, y) \in \boldsymbol{\mathcal{P}} / x \leq 0, y \leq 0, x^2 + y^2 = \delta^2\}$$

où $\square_{\mathbb{I}^2} E$ dénote la plus petite boîte contenant l'ensemble des valeurs de l'ensemble E. Notons que chacun de ces sous-domaines peut être réduit à l'ensemble vide.

Exemple 6.3.1 *Considérons les variables x et y de domaines respectifs $[-4,5]$ et $[-2,4]$ et la contrainte c :*

$$c : x^2 + y^2 = 25$$

La figure 7.3 montre que la décomposition sémantique découpe le domaine du point (x,y) en 3 sous-domaines : $[0,5] \times [0,4]$ réduit à $[3,5] \times [0,4]$, à l'aide de la projection de la contrainte c. De même, $[-4,0] \times [0,4]$ est réduit à $[-4,-3] \times [3,4]$. $[-4,0] \times [-2,0]$ est éliminé puisque la contrainte n'a pas de support dans ce domaine. Enfin, $[0,5] \times [-2,0]$ est réduit à $[4.5825,5] \times [-2,0]$.

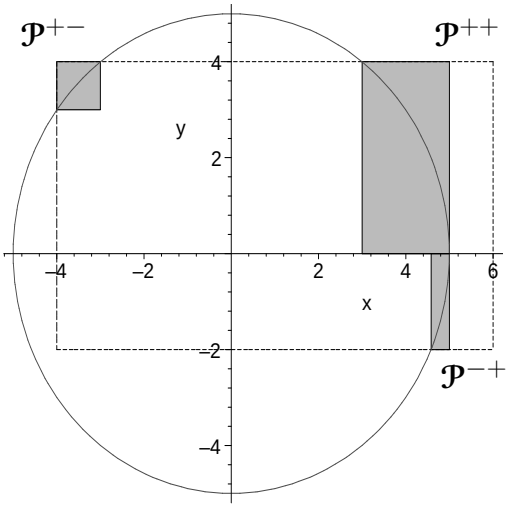


FIG. 6.3 – Illustration de la décomposition sémantique sur l'exemple 7.2.1.

La décomposition sémantique de $\mathcal{P}$ satisfait la propriété suivante :

Propriété 6.3.1 *Soit $f(x,y) = x^2 + y^2 - \delta^2$ et la contrainte définie par $f(x,y) = 0$ et Soit $s \in \{\mathcal{P}^{++}, \mathcal{P}^{-+}, \mathcal{P}^{+-}, \mathcal{P}^{--}\}$, tel que $s \neq \emptyset$. Alors la fonction f est monotone sur s.*

Preuve La monotonie est triviale puisque le vecteur gradient de f est $(2x, 2y)$ et que ses composantes sont de signe constant sur $\mathcal{P}^{++}$, $\mathcal{P}^{-+}$, $\mathcal{P}^{+-}$ et $\mathcal{P}^{--}$.

Comme f est monotone sur chacun des sous-domaines considérés, nous pouvons utiliser l'extension aux intervalles des fonctions de projection[26] associées à $f(x,y)$ pour calculer la plus petite boîte qui contient toutes les solutions de $c(x,y)$ sur le dit domaine.

Pour calculer une approximation linéaire des relations de distance, nous générons un ensemble d'inéquations qui encadrent chacun des espace solutions des sous-domaines monotones. Ces ensembles d'inéquations sont ensuite combinés pour obtenir une approximation du domaine complet. Nous présentons dans la suite les relaxations linéaires de $\mathcal{P}^{++}$, les autres se dérivant aisément par symétrie.

La section 3.3 montre comment obtenir une relaxation de $\mathcal{P}^{++}$ tandis que la section 3.4 montre comment combiner les approximations des sous-domaines monotones pour obtenir une relaxation du domaine $\mathcal{P}$.

6.3.3 Relaxation des contraintes de distance

Pour définir une approximation linéaire de l'espace des solutions, on distingue les relations de distance suivantes :

Distance exacte	c :	$x^2 + y^2 = \delta^2$
Distance minimale	$\underline{c}$:	$x^2 + y^2 \geq \delta^2$
Distance maximale	$\overline{c}$:	$x^2 + y^2 \leq \delta^2$

La contrainte c étant la conjonction des contraintes $\underline{c}$ et $\overline{c}$, le problème se ramène à trouver les relaxations de $\underline{c}$ et $\overline{c}$. Les relaxations de c seront définies par l'union de ces estimations.

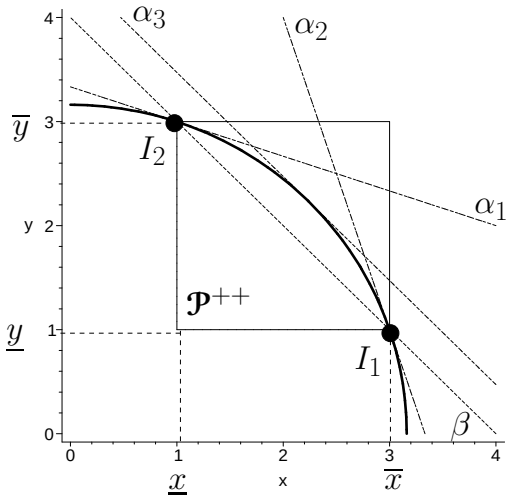


FIG. 6.4 – Relaxation de c sur $\mathcal{P}^{++}$.

Les relaxations de $\underline{c}$ et $\overline{c}$ sont calculées en considérant les points d'intersection entre le cercle défini par c et la boîte $\mathcal{P}^{++}$. La figure 6.4 montre que ces points, que l'on définit comme étant les *points d'intersection* de $\mathcal{P}^{++}$, sont $I_1(\overline{x}, \underline{y})$ et $I_2(\underline{x}, \overline{y})$.

De manière triviale, les tangentes en un point de l'arc de cercle $\widehat{I_1 I_2}$ définissent une surestimation des solutions de $\overline{c}$. De même, la corde $[I_1 I_2]$ définit une sous-estimation des solutions de $\underline{c}$.

Plus formellement, les propositions 6.3.1 et 6.3.2 définissent l'ensemble de sous-estimations de $\underline{c}$ et l'ensemble des surestimations de $\overline{c}$ sur le sous-domaine $\mathcal{P}^{++}$.

L'ensemble $\underline{\mathcal{L}}(\underline{c}, \mathcal{P}^{++})$ des sous-estimations de $\underline{c}$ correspondent au demi-plan délimité par le segment $[I_1 I_2]$ et qui ne contient pas le point $(0, 0)$.

Proposition 6.3.1 (Sous-estimations) *Considérons la contrainte* $\underline{c}$: $x^2 + y^2 \geq \delta^2$, *avec* $(x, y) \in \mathcal{P}^{++} = [\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$.

Soit $\underline{\mathcal{L}}(\underline{c}, \mathcal{P}^{++})$ *l'ensemble défini par la relation suivante :*

$$\beta : \quad (\overline{x} - \underline{x})y + (\underline{y} - \overline{y})x + \underline{x}\underline{y} - \overline{xy} \quad \geq 0$$

Alors, $\underline{\mathcal{L}}(\underline{c}, \mathcal{P}^{++})$ *est une sous-estimation linéaire de* $\underline{c}$.

L'ensemble $\overline{\mathcal{L}}(\overline{c}, \mathcal{P}^{++})$ des surestimations de $\overline{c}$ correspond à l'intersections des demi-plans contenant le point $(0, 0)$ et délimités par trois tangentes en I_1, en I_2 et au milieu de l'arc de cercle $\widehat{I_1 I_2}$.

Proposition 6.3.2 (Surestimations) *Considérons la contrainte* $\overline{c} : x^2 + y^2 \leq \delta^2$, *avec* $(x, y) \in \mathcal{P}^{++} = [\underline{x}, \overline{x}] \times [\underline{y}, \overline{y}]$.

Soit $\overline{\mathcal{L}}(\overline{c}, \mathcal{P}^{++})$, *l'ensemble défini par les relations suivantes :*

$$\begin{cases} \alpha_1 : & \overline{y}y + \underline{x}x - \delta^2 & \leq 0 \\ \alpha_2 : & \underline{y}y + \overline{x}x - \delta^2 & \leq 0 \\ \alpha_3 : & (\overline{x} - \underline{x})y + (\underline{y} - \overline{y})x - \gamma & \leq 0 \end{cases}$$

où $\gamma = \delta\sqrt{(\overline{x} - \underline{x})^2 + (\overline{y} - \underline{y})^2}$.
Alors, $\overline{\mathcal{L}}(\overline{c}, \mathcal{P}^{++})$ *est une surestimation linéaire de* $\overline{c}$.

Les surestimations et les sous-estimations de $\mathcal{P}^{+-}$, $\mathcal{P}^{-+}$ et $\mathcal{P}^{--}$ sont dérivées de $\mathcal{P}^{++}$ par symétrie. Les relaxations complètes de c, $\underline{c}$ et $\overline{c}$ sur les sous-domaine S sont alors définies par :

$$\begin{cases} \overline{\mathcal{L}}(c, S) = & \overline{\mathcal{L}}(\overline{c}, S) \\ \underline{\mathcal{L}}(c, S) & \underline{\mathcal{L}}(\underline{c}, S) \\ \mathcal{L}(c, S) = & \underline{\mathcal{L}}(c, S) \cup \overline{\mathcal{L}}(c, S) \end{cases}$$

La section suivante montre comment combiner ces approximations linéaires pour obtenir $\mathcal{L}(c, \mathcal{P})$, une approximation de c sur le domaine initial $\mathcal{P}$.

6.3.4 Combinaison des approximations

Les tangentes au cercle défini par c surestiment les sous-ensembles de solutions contenus par chacun les sous-domaines. Par conséquent, l'ensemble des surestimations de $\mathcal{P}$ est défini par l'union des surestimations de ses sous-domaines :

$$\overline{\mathcal{L}}(c, \mathcal{P}) = \overline{\mathcal{L}}(c, \mathcal{P}^{++}) \cup \overline{\mathcal{L}}(c, \mathcal{P}^{-+}) \cup \overline{\mathcal{L}}(c, \mathcal{P}^{+-}) \cup \overline{\mathcal{L}}(c, \mathcal{P}^{--})$$

La combinaison des sous-estimations des sous-domaines est plus compliquée ; l'union de ces approximations n'étant pas convexe en général.

Soit k le nombre de sous-domaines monotones non vides de $\mathcal{P}$. Il est évident que pour $k = 4$ il n'existe aucune sous-estimation convexe. Le problème se ramène donc à combiner les sous-estimations de c sur k sous-domaines monotones avec $k = 2$ ou $k = 3$. Il est facile de montrer que pour $k = 2$, les 2 sous-domaines sont contenus dans deux quadrants contigus.

Dans tous les cas, $\underline{\mathcal{L}}(c, \mathcal{P})$ doit être une sous-estimation de $\underline{\mathcal{L}}(c, S)$ pour tout sous-domaine S de $\mathcal{P}$. Considérons tous les demi-plans définis par les segments entre chaque couples de *points*

d'intersection. La sous-estimation globale est délimitée par le segment qui sous-estime l'ensemble de ces demi-plans, dans le domaine considéré.

Ce segment (ou ce plan en dimension 3) peut être déterminé en calculant l'enveloppe convexe de l'ensemble des *points d'intersection*. Dans le plan, on peut toutefois déterminer plus simplement ce segment en triant l'ensemble des points d'intersections par angle croissant par rapport à un point choisi dans un quadrant ne contenant pas de solutions. $\underline{\mathcal{L}}(c, \mathcal{P})$ est alors défini par le premier et le dernier point de cet ensemble trié.

La figure 6.5 montre un exemple d'une telle combinaison pour $k = 3$.

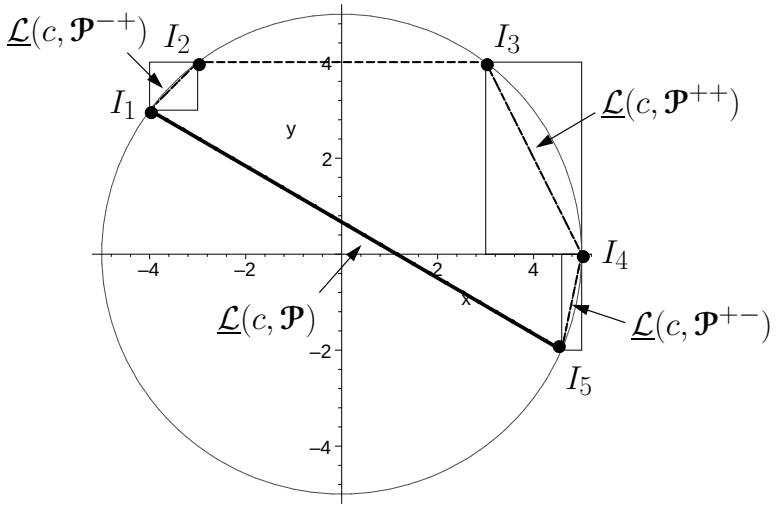


FIG. 6.5 – Sous-estimations de trois sous-domaines.

Dans la section suivante, nous comparons les systèmes linéaires engendrés par la Quad avec ceux générés par QuadDist.

6.4 Taille des programmes linéaires générés

Dans le cas général, c'est-à-dire pour des contraintes de distance dans un espace de dimension p, le nombre de contraintes générées par QuadDist est exponentiel. En effet, le nombre de contraintes linéaires générées est :

$$(p + 1)2^p m$$

où $(p+1)$ est le nombre de contraintes linéaires engendrées sur un sous-domaine monotone, 2^p est le nombre maximal de sous-domaines engendrés par la décomposition sémantique et m est le nombre de contraintes de distance du système initial. Toutefois, rappelons que QuadDist a été conçu pour résoudre des systèmes de contraintes de distance dans le plan et dans l'espace. Dans ce cas, le nombre de contraintes générées par QuadDist est tout à fait raisonnable.

Nous comparons dans ce chapitre la taille des programmes linéaires engendrés par QuadDist à ceux générés par Quad en dimension 2 et 3.

Considérons un CSP constitué de m contraintes de distance mettant en jeu n points.

Nombre de variables Quad utilise la forme développée des équations de distance en dimension 2 :

$$x_i^2 + x_j^2 + y_i^2 + y_j^2 + 2x_ix_j + 2y_iy_j = \delta_{ij}^2$$

et ajoute une variable supplémentaire pour chaque terme carré et pour chaque terme produit, soit $2n + 2m$ variables ajoutées aux $2n$ variables initiales et donc un total de $(4n + 2m)$ variables.

QuadDist n'introduit pas de variables additionnelles, donc le nombre de variables des systèmes linéaires engendrés est égal à $2n$.

Le nombre de variables des programmes linéaires engendrés par QuadDist est donc toujours plus petit.

Nombre de contraintes linéaires Au pire des cas, Quad engendre 3 inéquations linéaires pour chaque terme carré et 4 inéquations pour chaque terme produit, soit au total $C_Q = 4 \times 2m + 3 \times 2n = 8m + 6n$.

QuadDist calcule au pire des cas 3 surestimations sur chacun des 4 sous-domaines monotones et aucune sous-estimation. Le nombre de contraintes linéaires engendrées est donc 12 par contraintes, soit $C_{QD} = 12m$ contraintes linéaires.

Si le graphe de contraintes ne contient pas de noeuds isolés, c'est-à-dire si $n > m$, on a $C_Q - C_{QD} = 6n - 4m > 2m$, alors le nombre de contraintes engendrées par QuadDist est inférieur au nombre de contraintes générées par Quad. En dimension 3, on peut montrer de manière analogue que $C_Q - C_{QD} > 0$ si et seulement si $m < 9/20n$. Autrement dit, QuadDist génère toujours moins de contraintes linéaires que Quad pour des graphes de contraintes ayant une densité moyenne inférieure à 9/20.

6.5 Résultats expérimentaux

Nous avons comparé les temps de résolution de QuadDist avec et ceux de la Quad sur différents problèmes de contraintes de distance :

- Le problème du pentagone(penta1, penta2 et penta3) ainsi que sur une extension (ext-pentaN) de ce problème (Ces problèmes sont décrits dans la section 7.4.1).
- *octohedron0*, *octohedron1*, *octohedron2* sont 3 instanciation du problème de la molécule décrite dans [39].
- *minimal* et *minass* sont deux problèmes de contraintes géométriques décrites dans [59].
- *ponts0*, *ponts1* et *ponts2* sont 3 instances du problème des ponts décrit dans le chapitre 5. Ces 3 instances diffèrent sur le nombre de solutions, et ont été construite à partir d'une solution du système n prenant une boîte de plus en plus grande autour de cette solution. *ponts0* (resp. *ponts1*, *ponts2*) a 1 solution (resp. 15 solutions et 128 solutions).
- *robot2D* est le problème du robot plan décrit dans l'article [45], déjà mentionné dans l'introduction de cette thèse (voir figure 7.7 page 85). *viorel2* est une variante de ce problème (même structure, seules les distances changent).

Le tableau 7.8 récapitule les résultats obtenus sur un PC muni d'un processeur Pentium IV à 2Ghz avec 256Mo de RAM. Pour chacun des solveurs, on affiche les temps de résolution[1] en secondes (t) et le nombre d'appels au simplexe (S).

[1] C'est-à-dire recherche de toutes les solutions avec une précision de 1e-8

pb	Quad		QuadDist		Quad/QuadDist	
	S_Q	t_Q(s)	S_{QD}	t_{QD}(s)	S_{QD}/S_{QD}	t_Q/t_{QD}
dhingra	2263	10.730	1080	0.82	**2.09**	**13.08**
octohedron0	1542	6.282	6156	3.01	.25	**2.08**
octohedron1	1542	6.278	6156	2.94	.25	**2.13**
octohedron2	4028	15.212	14436	5.09	.27	**2.98**
minass	4846	16.104	2916	1.33	**1.66**	**12.10**
minimal	1188	1.886	1146	0.35	**1.03**	**5.38**
pentagone1	96	0.188	16	0.01	**6.00**	**18.80**
pentagone2	816	1.306	1008	0.29	.80	**4.50**
ext-penta1	95616	678.732	3356	2.41	**28.49**	**281.63**
ext-penta2	139960	1091.779	69956	39.96	**2.00**	**27.32**
ponts0	677	11.467	72	0.14	**9.40**	**81.90**
ponts1	792742	13028.784	171826	195.57	**4.61**	**66.61**
ponts2	1965649	36000.477	587398	703.50	**3.34**	**51.17**
robot2D	1254	1.845	1104	0.50	**1.13**	**3.69**
viorel2	9992	41.644	5928	3.99	**1.68**	**10.43**

FIG. 6.6 – Comparaison des temps de calculs pour Quad et pour QuadDist

	% reduction	iterations
Quad	56.45	402.23
QuadDist	56.63	356.67

TAB. 6.1 – Réduction des domaines et nombre d'itérations de Quad en comparaison avec QuadDist

On peut remarquer que pour l'ensemble de ces problèmes, QuadDist améliore significativement les performances de la Quad. Par exemple, pour le problème *ext-penta1*, QuadDist est 280 fois plus rapide que Quad. Le nombre d'appels au simplexe est également réduit, ce qui montre que QuadDist converge plus vite vers le point fixe.

6.5.1 Problèmes aléatoires

En nous alignant sur la méthodologie utilisée dans [38] et *KB03*, nous avons générés des instances de systèmes satisfiables de contraintes de distance avec un nombre de points n variant de 3 à 20 et un graphe de distance complet.

Pour chaque valeur de n, 25 problèmes ont été générés en prenant un ensemble aléatoires de points dans le plan. Les contraintes de distance sont données par les distances entre chaque paire de points. Les domaines sont générés en prenant une boîte de taille aléatoire autour des points. Les performances moyennes de Quad et de *QuadDist* pour chaque ensemble de problèmes sont reportées sur la figure 6.7.

Les deux algorithmes ont été paramètrés pour obtenir des boîtes avec une précision de 10^{-6} sur les bornes. Autrement dit, l'itération de relaxation+filtrage s'arrête lorsque la réduction globale des domaines est inférieure à 10^{-6}. Les calculs ont été effectués sous Linux, sur un PC équipé d'un processeur Pentium IV cadencé à 2.2Ghz avec 512Mo de mémoire.

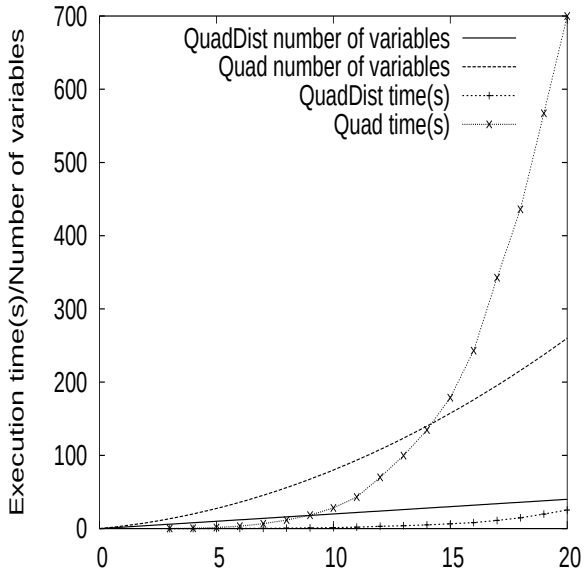


FIG. 6.7 – Temps d'execution par rapport au nombre de variables des systèmes linéaires engendrés

Sur ces instances aléatoires, on remarque que les deux algorithmes calculent pratiquement la même réduction des domaines, mais QuadDist requiert moins d'itérations pour approximer l'espace des solutions (voir table 6.1). En revanche, QuadDist est bien plus rapide que Quad : par exemple QuadDist requiert moins de 30s en moyenne pour filtrer un problème à 20 points, alors que Quad demande plus de 700s.

Troisième partie

Contribution à la résolution de CSPs continus

Chapitre 7

Décomposition sémantique pour les contraintes de distance

Les solveurs de systèmes de contraintes combinent des techniques de filtrage avec des techniques de recherche systématiques qui ne tiennent pas compte de la sémantique des contraintes. Nous proposons dans ce chapitre une technique de décomposition de domaines guidée par la sémantique des contraintes de distance. Cette décomposition sémantique, que nous nommerons SDD (Semantic Domain Decomposition), isole les sous-espaces disjoints dans l'espace de recherche. Les domaines des coordonnées d'un même point sont décomposés et réduits par la SDD en utilisant les propriétés de monotonie et de convexité des contraintes de distance. Nous montrons comment cette technique peut être combinée avec différents filtrages. L'apport de la SDD est mis en évidence sur différents CSPs constitués d'équations de distance.

7.1 Introduction

La satisfaction d'un système de contraintes numériques (ou NCSP[1]) est un problème qui a de nombreuses applications dans les sciences de l'ingénieur. Un NCSP est défini par un ensemble $\mathcal{X}$ de variables, un ensemble $\mathcal{D}$ de domaines associés à ces variables, ainsi qu'un ensemble $\mathcal{C}$ de contraintes constitué d'égalités et d'inégalités entre des fonctions numériques impliquant ces variables. La représentation des domaines par des intervalles a permis une adaptation des outils existants en programmation par contraintes sur des domaines finis. Plus particulièrement, les techniques de consistances locales [84], comme la kB-consistance [79, 80] ou la Box-consistance [99] calculent une approximation extérieure de l'ensemble des solutions.

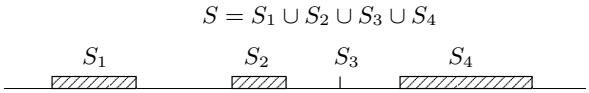


Fig. 7.1 – Disjonctions dans l'espace des solutions

Pour isoler les solutions ponctuelles du système, on utilise des méthodes d'énumération, comme la bisection, en combinaison avec ces techniques de filtrage local. La plupart des sol-

[1]Numerical Constraint Solving Problem

veurs de contraintes mettent en œuvre ces techniques de manière systématique, souvent sans tenir compte de la sémantique du problème. C'est notamment le cas pour la résolution de contraintes géométriques où les points sont représentés par un ensemble de variables traitées de manière indépendante. D'autre part, l'utilisation des propriétés des contraintes, comme la monotonie ou la convexité, améliore l'efficacité des techniques de résolution usuelles [21].

Nous proposons dans ce chapitre une stratégie de recherche qui s'appuie sur la sémantique des contraintes. L'intérêt de cette stratégie est illustrée sur des problèmes de satisfaction de contraintes de distance.

Différentes techniques spécifiques de filtrage ont été proposées pour la résolution de systèmes d'équations de distance, notamment les travaux de Lebbah *et al* qui introduisent une méthode de filtrage global nommée *Quad* et spécifiquement adaptée aux contraintes quadratiques [76, 89]. On peut citer également les travaux de Markot et Csendes qui présentent une technique basée sur une représentation des domaines par des polytopes convexes [85, 115, 86]. Cette technique, nommée *Method of "Active Areas"*, a été réintroduite par Heusch sous la forme d'une contrainte globale [51].

L'approche proposée ici est complémentaire à ces filtrages dans la mesure où elle guide la stratégie de recherche. Son objectif est d'identifier des points de choix les plus plus pertinents. C'est une approche alternative aux techniques de décomposition structurelle du graphe de contraintes introduites par Jermann *et al* [62, 60, 61]. En effet, cette dernière approche utilise la structure des systèmes de contraintes pour décomposer en sous-systèmes plus simples alors que nous proposons d'utiliser les propriétés des contraintes pour décomposer les domaines.

La section suivante décrit de manière informelle le principe de notre stratégie de recherche guidée par la sémantique des contraintes de distance.

7.2 Description informelle

Les méthodes de résolution de CSPs numériques sont souvent basées sur une approche de type Branch & Prune. Basiquement, l'idée est de découper le domaine d'une variable afin de diviser le problème en deux sous-problèmes (Branch). Chaque sous-problème est alors réduit ou éliminé en utilisant une méthode de filtrage, par exemple une consistance locale. Puis, un nouveau domaine est choisi pour être découpé et le processus recommence jusqu'à ce que tous les domaines soient de taille inférieure à un certain paramètre de précision.

Notre décomposition se place en amont de la résolution proprement dite dans la mesure où elle divise l'espace de recherche en sous-espaces vérifiant de fortes propriétés (consistance locale et monotonie des contraintes) mais non nécessairement réduits à une solution ponctuelle.

7.2.1 Décomposition sémantique des domaines

Nous introduisons ici une technique de décomposition sémantique pour les contraintes de distance, nommée SDD (Semantic Domain Decomposition). Cette méthode découpe et réduit les domaines des points mis en jeu dans une contrainte de distance en utilisant ses propriétés

de monotonie. La forme canonique des équations de distance [2] $X^2 + Y^2 - D^2 = 0$ est utilisée pour isoler ces parties monotones sur $\mathbb{R}^+ \times \mathbb{R}^+$, $\mathbb{R}^+ \times \mathbb{R}^-$, $\mathbb{R}^- \times \mathbb{R}^+$ et $\mathbb{R}^- \times \mathbb{R}^-$.

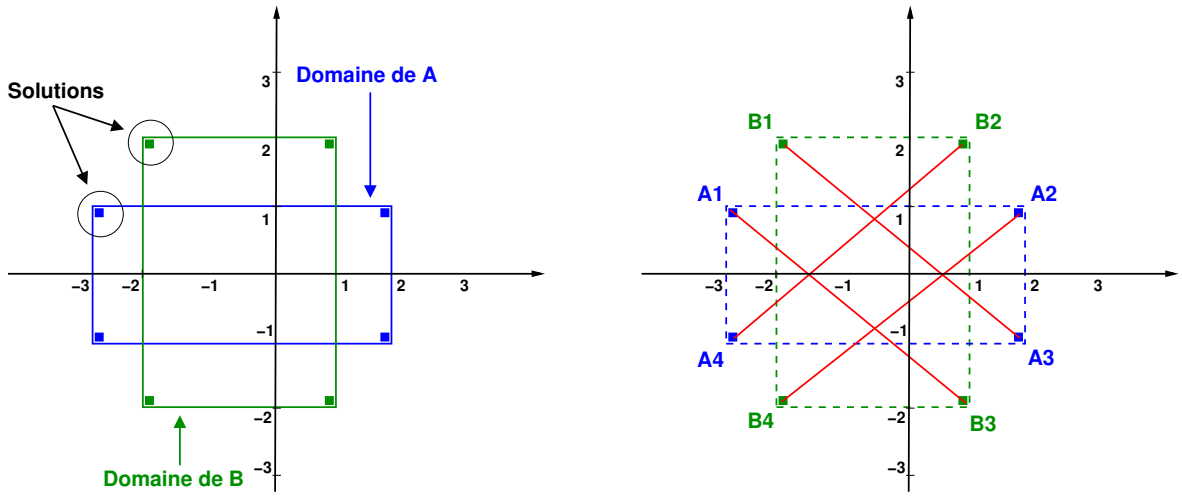


FIG. 7.2 – La figure de gauche montre les domaines de A et B après un filtrage par 2B-consistance. Les carrés pleins représentent les continuums de solutions. La figure de droite montre les 4 sous-domaines de A et ceux de B ainsi que leurs liaisons dans la micro-structure calculée par la SDD.

Un changement de variable linéaire permet de passer de la forme canonique à une contrainte et ainsi d'extraire les sous-domaines des points sur ces parties monotones. L'exemple suivant illustre les apports de SDD par rapport aux consistances locales :

Exemple 7.2.1 *Soient 2 points du plan $A(x_A, y_A)$ et $B(x_B, y_B)$ dont les domaines sont respectivement les boîtes $[-3, 2] \times [-1, 1]$ et $[-2, 1] \times [-2, 2]$. On cherche à déterminer les points A et B tels que la longueur du segment $[AB]$, notée D, soit dans l'intervalle $[4.95, 5]$. Le CSP correspondant est décrit par la contrainte $(x_A - x_B)^2 + (y_A - y_B)^2 = D^2$, avec $x_A \in [-3, 2]$, $x_B \in [-1, 1]$, $y_A \in [-2, 1]$, $y_B \in [-2, 2]$ et $D \in [4.95, 5]$.*

La 2B-consistance ne permet pas de réduire ces domaines ; la figure 7.2 (à gauche) montre que les continuums de solutions sont proches des bornes des domaines. À droite, la figure 7.2 montre que la SDD divise le CSP initial en 4 sous-CSP correspondant aux 4 sous-espaces de solutions locales. Ces sous-espaces sont modélisés par un graphe de sous-domaines dont les arêtes sont les segments A_1B_3, A_2B_4, A_3B_1 et A_4B_2.

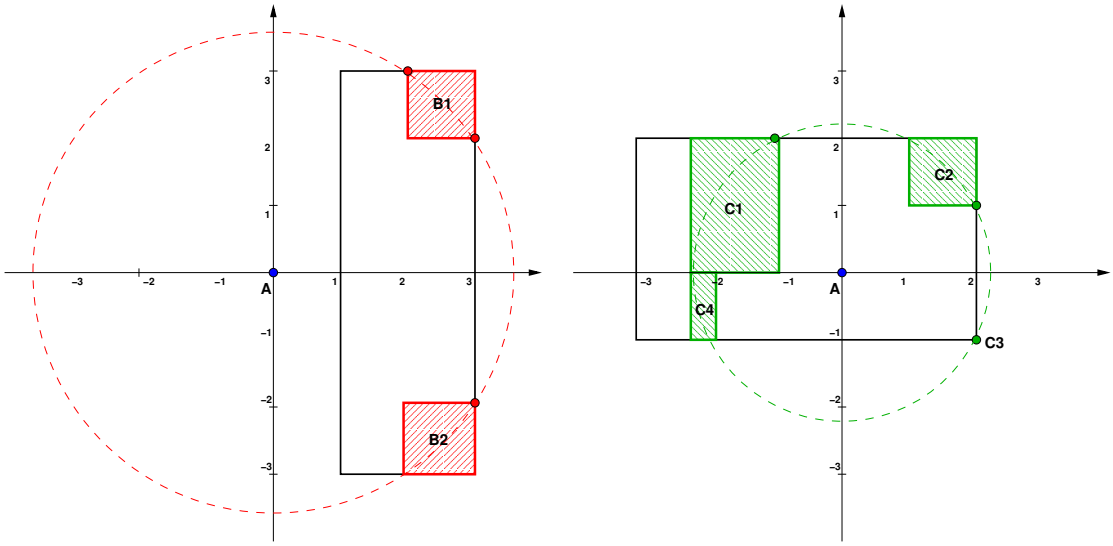
Notons qu'avec une précision de l'ordre de 0.05 la bisection combinée à la 2B permettrait d'obtenir un résultat similaire mais avec 2 étapes de division alors que la SDD ne nécessite qu'une étape. La combinaison 2B & bisection avec une petite précision conduirait à énumérer un grand nombre de points solutions du système.

Une approche basée sur les unions d'intervalles [15] permettrait également d'obtenir une décomposition similaire pour chacun des domaines mais sans la micro-structure. Sans ces liaisons, les 16 combinaisons de sous-domaines doivent être examinées afin de trouver les sous-espaces potentiellement porteurs de solutions.

[2] En dimension 2 pour simplifier la notation. La généralisation en dimension quelconque est triviale.

7.2.2 Décomposition sémantique d'un CSP

Dans cette section, nous montrons sur un exemple simple comment la SDD est propagée sur l'ensemble du système de contraintes. Le résultat de cette propagation est une décomposition du CSP initial en un ensemble de sous-domaines sur lesquels toutes les contraintes sont consistantes et monotones. Dans cet exemple notre algorithme de décomposition suffit à isoler rapidement les solutions du système.



(a) (b)

FIG. 7.3 – La figure 7.3(a) montre la décomposition du domaine de B en 2 sous-domaines B_1 et B_2, pour la contrainte c_1. La figure 7.3(b) montre la décomposition du domaine de C en 4 sous-domaines C_1, C_2, C_3 et C_4, pour la contrainte c_2.

Exemple 7.2.2 *Considérons dans le plan, 3 points $A(0,0)$, $B(x_B, y_B)$ et $C(x_C, y_C)$ où le point P_1 est fixé à l'origine du repère cartésien. Les domaines initiaux de x_B, y_B, x_C et y_C sont respectivement $[1,3]$, $[-3,3]$, $[-3,2]$ et $[-1,2]$. Les distances entre les points sont données par : $AB^2 = 13$,$AC^2 = 5$ et $BC^2 = 10$. Le CSP correspondant est le suivant :*

- $\mathcal{X} = \{x_B, y_B, x_C, y_C\}$.
- $\mathcal{D} = \{[1,3], [-3,3], [-3,2], [-1,2]\}$.
- $\mathcal{C} = \begin{cases} c_1 : & x_B^2 + y_B^2 = 13 \\ c_2 : & x_C^2 + y_C^2 = 5 \\ c_3 : & (x_B - x_C)^2 + (y_B - y_C)^2 = 10 \end{cases}$

Les 3 solutions de ce CSP sont représentées graphiquement sur la figure 7.4.

La SDD appliquée à la contrainte c_1 décompose le domaine initial de B en 2 sous-domaines B_1 et B_2. De même, la contrainte c_2 permet à la SDD de décomposer le domaine de C en 4 sous-domaines C_1, C_2, C_3 et C_4. La figure 7.3 montre le résultat de cette décomposition. Le traitement de la contrainte c_3 nous conduit à examiner les 8 combinaisons de sous-domaines de B et C :

- B_1C_4 et B_2C_4 sont immédiatement éliminées car C_4 n'a de support ni dans B_1, ni dans B_2.

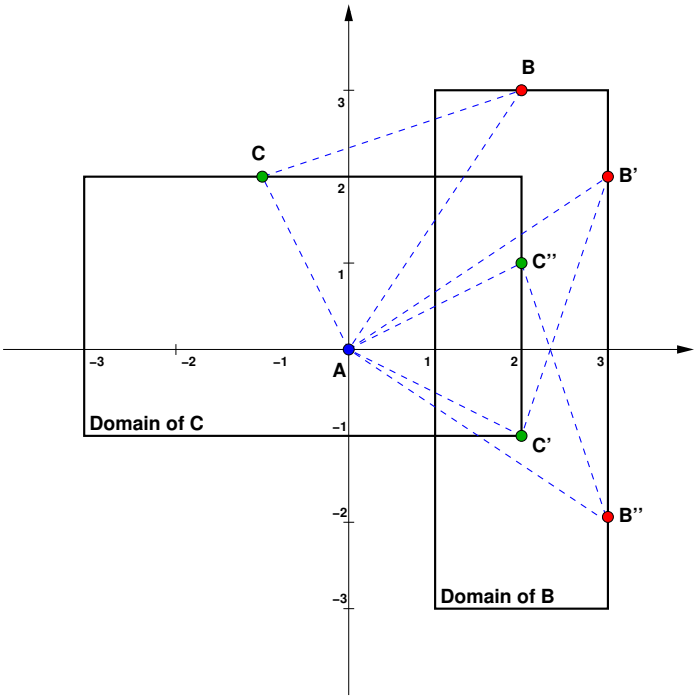


FIG. 7.4 – Les 3 solutions du CSP de l'exemple 7.2.2 sont les triangles ABC, $AB'C'$ et $AB''C''$.

- De même pour B_2C_3 et B_2C_1, car B_2 n'a de support ni dans C_3, ni dans C_1.
- Une consistance locale appliquée à B_1C_1, B_1C_3 et B_2C_2 suffit à isoler les 3 solutions du système.

La suite de ce chapitre est organisée de la manière suivante : la section 7.3 détaille la décomposition sémantique des domaines et l'algorithme de décomposition de CSP. Enfin, la section 8.4 montre que la SDD améliore les performances des méthodes traditionnelles (consistance locale couplée avec la bisection) pour différentes variantes du classique problème du pentagone, ainsi que pour un problème classique de robotique.

7.3 Décomposition sémantique d'un CSP

Les solveurs de contraintes utilisent habituellement une technique de bisection qui consiste à découper un domaine en deux parties généralement de même taille. Différentes heuristiques (taille des domaines, variables contiguës, ...) peuvent être utilisées pour sélectionner le domaine de la variable à bisecter. C'est une méthode systématique qui ne prend pas en compte les informations spécifiques des contraintes considérées.

Nous proposons une stratégie de résolution qui tire profit des propriétés spécifiques des contraintes de distance. Les principales caractéristiques de cette méthode sont :

- Un découpage simultané de toutes les variables liées par une contrainte.
- Une stratégie de choix de point de coupe qui exploite les propriétés sémantiques des contraintes (monotonie et convexité).

Cette stratégie améliore les performances des solveurs de contraintes basées sur des consistances locales (*c.f.* expérimentations dans la dernière section de ce chapitre).

Dans cette section nous décrivons plus formellement notre algorithme de décomposition. La section 7.3.2 détaille la réécriture du CSP initial avant l'application de notre stratégie de recherche. Dans la section 7.3.3, nous verrons que le schéma général de cet algorithme repose non sur un découpage systématique des domaines mais guidé par la sémantique des contraintes.

7.3.1 Décomposition sémantique d'une contrainte de distance

Dans le chapitre 6, nous avons défini la décomposition sémantique des contraintes de distance. Plus formellement,

Définition 7.3.1 *Considérons la contrainte* $c : x^2 + y^2 = \delta^2$ *avec* $\mathcal{P}(x, y) \in \boldsymbol{\mathcal{P}} = \boldsymbol{x} \times \boldsymbol{y}$ *et* $\delta > 0$. *On définit* $\mathsf{SDD}(c, \boldsymbol{\mathcal{P}})$ *par l'ensemble des sous-domaines suivants :*

$$\boldsymbol{\mathcal{P}}^{++} = \square_{\mathbb{I}^2}\{(x, y) \in \boldsymbol{\mathcal{P}} / x \geq 0, y \geq 0, x^2 + y^2 = \delta^2\}$$

$$\boldsymbol{\mathcal{P}}^{-+} = \square_{\mathbb{I}^2}\{(x, y) \in \boldsymbol{\mathcal{P}} / x \geq 0, y \leq 0, x^2 + y^2 = \delta^2\}$$

$$\boldsymbol{\mathcal{P}}^{+-} = \square_{\mathbb{I}^2}\{(x, y) \in \boldsymbol{\mathcal{P}} / x \leq 0, y \geq 0, x^2 + y^2 = \delta^2\}$$

$$\boldsymbol{\mathcal{P}}^{--} = \square_{\mathbb{I}^2}\{(x, y) \in \boldsymbol{\mathcal{P}} / x \leq 0, y \leq 0, x^2 + y^2 = \delta^2\}$$

où $\square_{\mathbb{I}^2} E$ *dénote le plus petit rectangle contenant l'ensemble des valeurs de l'ensemble* E. *Notons que chacun de ces sous-domaines peut être réduit à l'ensemble vide.*

La fonction SDD (*c.f.* Fonction 1) réalise cette décomposition sémantique en considérant une contrainte de distance sous forme canonique C et le domaine $\boldsymbol{X} = \boldsymbol{x} \times \boldsymbol{y}$ du vecteur associé à C. La réduction des sous-domaines (ligne 7) est effectuée par la fonction $\mathsf{Narrow}(\boldsymbol{X}, C)$ qui utilise les fonctions de projection suivantes :

$$\boldsymbol{x} \leftarrow \boldsymbol{x} \cap SQRT(\boldsymbol{\delta}^2 - \boldsymbol{x}^2)$$

$$\boldsymbol{y} \leftarrow \boldsymbol{y} \cap SQRT(\boldsymbol{\delta}^2 - \boldsymbol{y}^2)$$

où $SQRT$ est l'extension aux intervalles[26] de la fonction $\sqrt{}$. Les sous-domaines non réduits à l'ensemble vide sont ajoutés à l'ensemble S, destiné à contenir les sous-domaines résultant de cette décomposition (ligne 9).

Cette fonction dont la complexité temporelle est constante renvoie donc l'ensemble des sous-domaines décrits par la proposition 6.3.1.

Notons qu'en dimension p le nombre maximal de sous-domaines engendrés par cette décomposition est exponentiel et égal à 2^p. Néanmoins cette méthode est tout à fait adaptée en dimension faible (2 et 3), où les applications faisant intervenir des contraintes de distance sont les plus nombreuses.

Notons également qu'un sous-domaine s issu de la décomposition sémantique ne peut être décomposé à plusieurs reprises, ce qui se traduit par la propriété suivante :

Propriété 7.3.1 (Point fixe) *Soit* $s \in \mathsf{SDD}(c, \boldsymbol{X})$ *alors* $\mathsf{SDD}(c, s) = s$.

Function 1 $\mathsf{SDD}(C, \boldsymbol{X})$

1: $\boldsymbol{X}_1 \leftarrow \{(x,y) \in \boldsymbol{X} : x \geq 0 \wedge y \geq 0\}$
2: $\boldsymbol{X}_2 \leftarrow \{(x,y) \in D : x \leq 0 \wedge y \geq 0\}$
3: $\boldsymbol{X}_3 \leftarrow \{(x,y) \in D : x \leq 0 \wedge y \leq 0\}$
4: $\boldsymbol{X}_4 \leftarrow \{(x,y) \in D : x \geq 0 \wedge y \leq 0\}$
5: $S \leftarrow \emptyset$
6: **for all** i such that $1 \leq i \leq 4$ **do**
7: $\quad \boldsymbol{X}_i \leftarrow \mathsf{Narrow}(\boldsymbol{X}_i, C)$
8: $\quad$ **if** $\boldsymbol{X}_i \neq \emptyset$ **then**
9: $\quad\quad S \leftarrow S \cup \{\boldsymbol{X}_i\}$
10: $\quad$ **end if**
11: **end for**
12: **return** S

7.3.2 Réécriture du CSP initial

Considérons un CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ constitué exclusivement de m contraintes de distance de la forme[3] :

$$c_k : (x_{j_k} - x_{i_k})^2 + (y_{j_k} - y_{i_k})^2 = \delta_k^2$$

La réécriture du CSP initial consiste à mettre toutes les contraintes sous forme canonique. Pour chaque contrainte c_k on introduit deux variables additionnelles X_k et Y_k correspondant aux coordonnées du vecteur $\overrightarrow{P_{i_k}P_{j_k}}$. Les domaines $\boldsymbol{V}_k$ des vecteurs (X_k, Y_k) sont calculés en utilisant l'arithmétique des intervalles. La réécriture du système consiste à ajouter les contraintes de changement de repère et à remplacer c_k par sa forme canonique :

$$\begin{cases} c'_k : X_k^2 + Y_k^2 = \delta_k^2 \\ X_k = x_{j_k} - x_{i_k} \\ Y_k = y_{j_k} - y_{i_k} \end{cases}$$

L'ensemble des domaines est donc de la forme :

$$\mathcal{D} = \{\boldsymbol{V}_1, \ldots, \boldsymbol{V}_m, \boldsymbol{\mathcal{P}}_1, \ldots, \boldsymbol{\mathcal{P}}_n\}$$

où $\boldsymbol{\mathcal{P}}_i$ est le domaine du point P_i avec $1 \leq i \leq n$, et $\boldsymbol{V}_k$ est le domaine du vecteur (X_k, Y_k) associé à la contrainte c'_k avec $1 \leq k \leq m$.

Les contraintes c'_k étant sous forme canonique, les domaines $\boldsymbol{V}_k$ des vecteurs (X_k, Y_k) pourront être décomposés en utilisant la technique de décomposition sémantique (*c.f* section 7.3.1).

7.3.3 Algorithme de décomposition sémantique

Notre algorithme de Décomposition Sémantique, nommé DS, considère un CSP (X, D, C) et calcule une décomposition du domaine initial D et un ensemble de sous-domaines sur lesquels :

– toutes les contraintes de C sont monotones

[3] Nous décrivons notre méthode en dimension 2, l'extension en dimension supérieure est triviale

– une propriété de consistance locale est vérifiée

Par exemple, soit le CSP initial $(\{x, y\}, \{\boldsymbol{x}, \boldsymbol{y}\}, C)$. DS va générer un ensemble de couples de la forme $\{\boldsymbol{x}^i, \boldsymbol{y}^j\}$ où $\boldsymbol{x}^i \subset \boldsymbol{x}$ et $\boldsymbol{y}^j \subset \boldsymbol{y}$ De plus, pour chacun de ces couples engendrés par DS, les CSP $(\{x, y\}, \{\boldsymbol{x}^i, \boldsymbol{y}^j\}, C)$ vérifient une propriété de consistance locale.

DS est donc paramétré par une fonction nommée Prune qui étant donné un CSP (X, D, C) renvoie l'ensemble des domaines obtenus par un filtrage utilisant une consistance locale(*e.g.* 2B-consistance[79, 80], Box-consistance[99]). Si le CSP ne vérifie pas la consistance locale associée à Prune, cette fonction renvoie un ensemble vide.

Le principe de DS (voir fonction 2) est sensiblement le même que celui d'un algorithme de recherche arborescente basé sur la bisection. La principale différence étant le fait que les domaines des (X_k, Y_k) sont décomposés en utilisant la décomposition sémantique décrite précédemment. Contrairement à la bisection dont le processus de décomposition ne s'arrête que lorsque tous les domaines sont suffisamment petits (de taille inférieure à un $\epsilon > 0$ donné), le point fixe est atteint lorsque tous les domaines on été considérés (voir propriété 7.3.1). Autrement dit, tous les domaines des vecteurs $\boldsymbol{V}_k$ vérifient la propriété $\boldsymbol{V}_k = \mathsf{SDD}(\boldsymbol{V}_k, c_k)$, ce qui signifie que $\boldsymbol{V}_k$ a déjà été décomposé par SDD.

Function 2 $\mathsf{DS}(\mathcal{X}, \mathcal{D}, \mathcal{C})$

1: $R \leftarrow \emptyset$
2: $Q \leftarrow \{\mathcal{D}\}$
3: **while** $Q \neq \emptyset$ **do**
4: Choose S from Q %% $S = \{\boldsymbol{V}_1, \ldots, \boldsymbol{V}_m, \mathcal{P}_1, \ldots, \mathcal{P}_n\}$
5: **if** $\boldsymbol{V}_k = \mathsf{SDD}(\boldsymbol{V}_k, c_k)$ for all $\boldsymbol{V}_k \in S$ **then**
6: $R \leftarrow R \cup \{S\}$
7: **else**
8: Choose $\boldsymbol{V}_k$ from S
9: Let $c_k \in C$ the constraint associated to $\boldsymbol{V}_k$.
10: **for all** $s \in \mathsf{SDD}(\boldsymbol{V}_k, c_k)$ **do**
11: $S' \leftarrow S[\boldsymbol{V}_k \leftarrow s]$ %% $S' = \{\boldsymbol{V}_1, \ldots, \boldsymbol{V}_{k-1}, s, \boldsymbol{V}_{k+1}, \ldots, \boldsymbol{V}_m, \mathcal{P}_1, \ldots, \mathcal{P}_n\}$
12: $S' \leftarrow \mathsf{Prune}(\mathcal{X}, S', \mathcal{C})$ %% Local filtering of the domain S'
13: **if** $S' \neq \emptyset$ **then**
14: $Q \leftarrow Q \cup \{S'\}$
15: **end if**
16: **end for**
17: **end if**
18: **end while**
19: **return** R

Plus précisément, DS utilise un ensemble Q contenant des sous-domaines du domaine initial et initialisé à $\{\mathcal{D}\}$ (ligne 2). S est alors extrait de l'ensemble Q et l'on vérifie si le point fixe est atteint (lignes 4 et 5).

Si c'est le cas, S est ajouté à R, l'ensemble destiné à contenir les sous-domaines engendrés par DS (ligne 6). Sinon, on choisit un vecteur dont le domaine $\boldsymbol{V}_k$ n'a pas encore été décomposé par SDD (ligne 8). Pour tous les sous-domaines engendrés par la décomposition sémantique du domaine du vecteur choisi (ligne 10), on ajoute à Q les sous-domaines de S correspondants

s'ils vérifient la propriété de consistance locale (lignes 10 à 15). L'appel à la fonction Prune utilise toutes les contraintes de $\mathcal{C}$ pour réduire les domaines de toutes les variables.

L'heuristique utilisée pour choisir le sous-domaine extrait de Q (ligne 4) est de choisir celui dont la décomposition est la plus avancée. Cela correspond à une recherche arborescente en profondeur d'abord, ce choix se fait donc en temps constant.

L'heuristique utilisée pour choisir le prochain domaine à décomposer (ligne 8) est de choisir celui dont la décomposition par SDD engendre un nombre de sous-domaines minimal et strictement supérieur à 1. Une recherche exhaustive du minimum est utilisée avec une complexité temporelle en $\mathcal{O}(m)$ où m est le nombre de vecteurs, c'est-à-dire de contraintes du système.

Nous allons maintenant montrer l'apport de notre algorithme de décomposition sur différentes expérimentations.

7.4 Résultats expérimentaux

Les sections 7.4.1 et 7.4.2 détaillent respectivement les problèmes du pentagone et de la cinématique directe d'un robot plan. Les résultats obtenus par DS sur ces problèmes sont détaillés dans la section 7.4.3, ainsi que la comparaison de ces résultats avec ceux obtenus avec la bisection.

7.4.1 Problème classique du pentagone

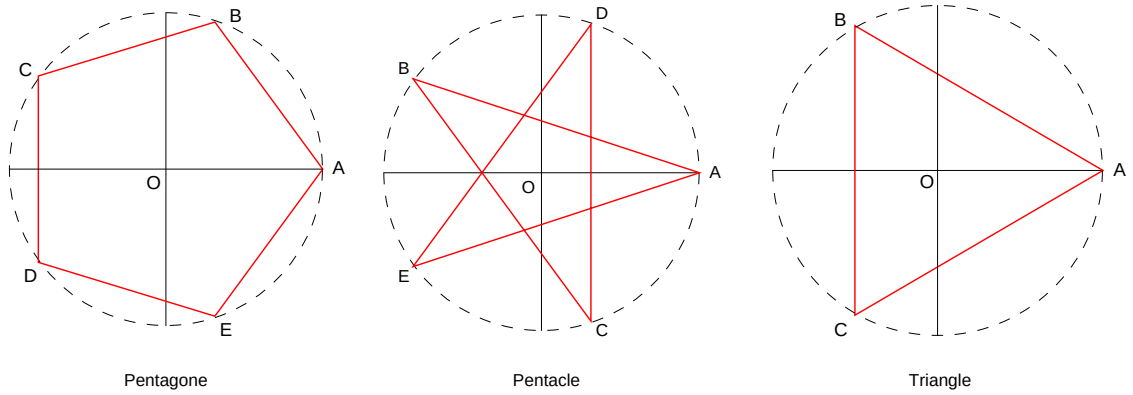


Fig. 7.5 – Solutions du problème du pentagone

Le problème du pentagone est un CSP bien connu dans la communauté de la programmation par contraintes sur les domaines continus. Il est souvent utilisé pour illustrer les problèmes que rencontrent les méthodes de filtrage local, lorsque l'espace solution est fractionné. Il s'agit d'un CSP géométrique en 2D, constitué de 5 points P_1,P_2,P_3,P_4, et P_5 sur le cercle unitaire, tels que les segments P_1P_2, P_2P_3, P_3P_4, P_4P_5, et P_1P_5 soient de longueur égale à d. L'un de ces 5 points est fixé au point de coordonnées (1,0) pour éviter que le nombre de solutions ne soit infini. Selon le choix de la distance d entre deux points consécutifs, on obtient 3 instances que nous nommerons *penta1*, *penta2* et *penta3* donnant lieu à 3 types de solutions (Fig. 7.5) :

1. *penta1* : Si $d = 2\sin(\frac{\pi}{5})$ il y a 2 solutions pour la combinaison $P_1P_2P_3P_4P_5$ formant un pentagone, $ABCDE$, $AEDCB$.
2. *penta2* : Si $d = 2\sin(\frac{2\pi}{5})$ il y a 2 solutions pour la combinaison $P_1P_2P_3P_4P_5$ formant un pentacle, $ABCDE$, $AEDCB$.
3. *penta3* : Si $d = 2\sin(\frac{2\pi}{3})$ il y a 10 solutions pour la combinaison $P_1P_2P_3P_4P_5$ formant un triangle, $ABCAB$,$ACBAC$, $ABCAC$, $ACBAB$, $ABABC$, $ACACB$, $ABCBC$, $ACBCB$, $ABACB$ et $ACABC$.

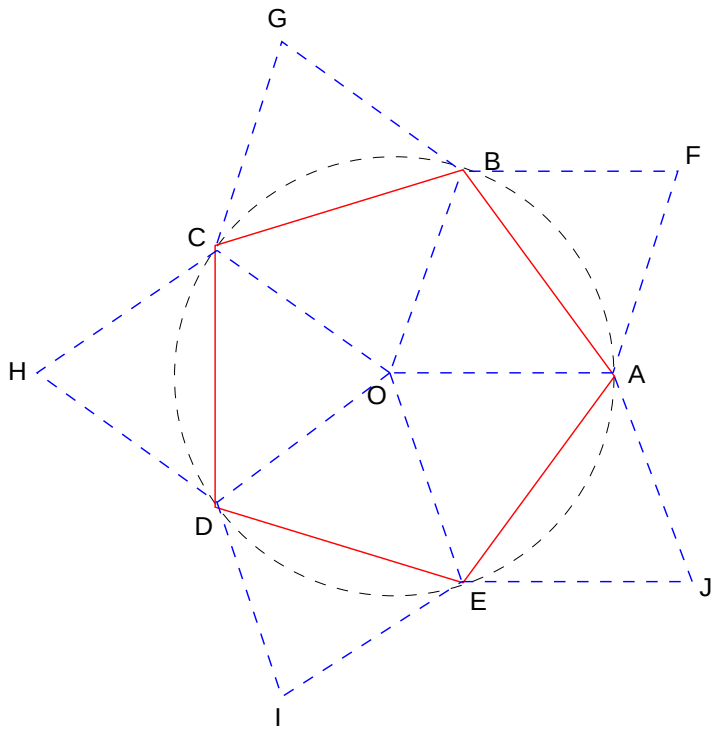


Fig. 7.6 – Ajout de contraintes dans le problème du pentagone

Pour compliquer un peu le problème, cinq points supplémentaires sont ajoutés au problème initial, un entre chaque arête à une distance de 1 de chaque extrémité (Fig. 7.6). Pour une solution du problème classique, cela engendre $2^5 = 32$ fois plus de solutions, soit 64 pour *penta1* et *penta2*, et 320 pour *penta3*. L'instance correspondant à l'extension de *penta1* (resp. *penta2*, *penta3*) étendue par ce procédé sera nommée *ext-penta1* (resp. *ext-penta2*, *ext-penta3*)

7.4.2 Cinématique directe d'un robot plan

Le manipulateur de type 3-RPR est formé de deux triangles : la base ABC et la plate-forme DEF reliés par 3 jambes AD, BE et CF (voir figure 7.7). Le problème de la cinématique directe est de trouver toutes les positions de la plate-forme compatibles avec les longueurs des jambes qui sont les données du problème. Ce problème possède 6 solutions réelles[45].

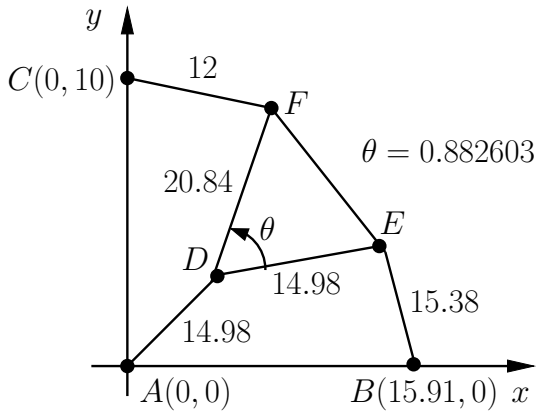


FIG. 7.7 – Un manipulateur planaire de type 3-RPR

Pb(#sol)	Bisection				DS			
	2B(10^{-10})		Box(10^{-10})		2B(10^{-10})		Box(10^{-10})	
	t(s)	#box	t(s)	#box	t(s)	#box	t(s)	#box
penta1(2)	0.02s	4	0.16s	4	0.03s	2	0.16s	2
penta2(2)	0.03s	100	0.07s	2	0.03s	2	0.03s	2
penta3(10)	0.04s	56	0.63s	192	0.03s	10	0.18s	10
ext-penta1(64)	1.47s	5128	3.02s	64	0.26s	64	2.71s	64
ext-penta2(64)	3457.32s	430798	1.91s	64	0.08s	16	0.52s	16
ext-penta3(320)	21.44s	87642	1705.45s	210398	1.12s	320	9.53s	320
robot2D(6)	7.77s	127	1.7s	12	0.26s	18	0.37s	20

FIG. 7.8 – Résultats expérimentaux

7.4.3 Résultats et analyse

Nous avons comparé les performances de DS avec celles de la bisection en combinant ces méthodes avec la 2B ou la Box avec une précision de 10^{-10}. Les expérimentations ont été réalisés en utilisant le solveur Ilog Solver 5.0 [54] sur un PC équipé d'un microprocesseur pentium IV à 2GhZ et 128Mo de mémoire.

Les résultats (voir tableau 7.8) montrent que pour les trois premiers exemples (penta1,penta2, penta3), les temps de toutes les approches sont de l'ordre du centième de secondes et la comparaison n'est pas significative. On note toutefois que DS génère une boîte par solution alors que la bisection génère de nombreuses boîtes parasites.

Pour *ext-penta3*, les performances de DS sont bien meilleures que celles d'un algorithme de recherche basé sur la bisection (le facteur de gain est compris entre 20 et 200). Pour *ext-penta2*, le gain de temps est également significatif mais certaines boîtes doivent encore être découpées pour isoler les solutions.

Pour robot2D, le gain de temps est également significatif mais quelques boîtes sans solutions sont générées, ce qui est dû aux faibles capacités de filtrage des consistances locales utilisées. Sur cet ensemble de tests préliminaires, les gains en performances et précision sont très encourageants.

Chapitre 8

Mind The Gaps : Une nouvelle stratégie de résolution pour les techniques de consistances

Les outils classiques pour résoudre des CSPs numériques sont basés sur un algorithme de type Branch&Prune, une énumération dichotomique des domaines alternée avec un filtrage par consistance. Dans la plupart des solveurs d'intervalles, l'étape de filtrage est réalisée par des consistances locales (Hull-Consistance [79, 80, 13], Box-consistance [14, 119, 26]) ou des consistances partielles (kB-consistances [79, 74], Bound-consistance [100]). Les algorithmes de filtrage associés identifient souvent des trous dans les domaines, *i.e.*des intervalles de valeurs inconsistantes strictement inclus dans les domaines. Pourtant ces trous sont utilisés uniquement dans le but de calculer un filtrage plus fort des domaines.

Dans ce chapitre, nous présentons un stratégie de recherche, nommée MindTheGaps, qui exploite les trous identifiés durant l'étape de filtrage. Les trous sont collectés avec un sur-coût minime et sont utilisés pour choisir la direction de coupe ainsi que pour définir les points de coupe dans le domaine sélectionné.

Découper le domaine en éliminant ces trous permet de réduire l'espace de recherche, ce qui n'est pas le cas des techniques de splitting classique comme la bisection. Cela aide le solveurs à éliminer certaines solutions redondantes et à isoler des solutions disjointes. Des résultats expérimentaux montrent que MindTheGaps améliore de manière significative les performances de l'algorithme de résolution classique.

8.1 Introduction

Les consistances locales pour des CSPs aux domaines finis ont été adaptées aux CSPs numériques. Les algorithmes de filtrage associés éliminent des domaines certaines valeurs inconsistantes, c'est-à-dire pour lesquelles il existe au moins une contrainte qui ne peut être vérifiée. En pratique, le filtrage est en général limité à la contraction des bornes des intervalles.

Les techniques classiques pour résoudre des CSPs numériques sont basées sur un algorithme de type Branch&Prune. Cet algorithme alterne filtrages et décompositions de domaines

jusqu'à ce qu'une précision donnée sur les domaines soit atteinte. L'étape de décomposition (ou *splitting*) sélectionne une variable et découpe son domaine en plusieurs parties. Les sous-problèmes générés sont alors résolus de manière indépendante. La technique la plus souvent utilisée est la bisection, qui coupe le domaine sélectionné en 2 parties de même taille. Diverses stratégies de sélection de domaines ont déjà été présentées dans la section 3.4.1 page 32. Parmi ces méthodes, celle qui est considérée comme la plus efficace est le Round Robin (RR), qui sélectionne les variables à tour de rôle.

Dans la plupart des solveurs d'intervalles, l'étape de filtrage est réalisée par des consistances locales (Hull-Consistance [79, 80, 13], Box-consistance [14, 119, 26]) ou des consistances partielles (kB-consistances [79, 74], Bound-consistance [100]). Les algorithmes de filtrage associés identifient souvent des trous dans les domaines, *i.e.*des intervalles de valeurs inconsistantes strictement inclus dans les domaines. Or, ces trous sont utilisés uniquement dans le but de calculer un filtrage plus fort des domaines.

Dans ce chapitre, nous présentons un stratégie de recherche, nommée MindTheGaps, qui exploite les trous identifiés durant l'étape de filtrage. Les trous sont collectés avec un sur-coût minime et sont utilisés pour sélectionner la direction de coupe ainsi que pour définir les points de coupe dans le domaine sélectionné. Si aucun trou n'a été identifié, l'étape de décomposition est réalisée en utilisant bisection combinée avec une heuristique standard de choix de variable.

Découper le domaine en éliminant ces trous permet de réduire l'espace de recherche, ce qui n'est pas le cas des techniques de splitting classique comme la bisection. Cela aide le solveurs à éliminer certaines solutions redondantes et à isoler des solutions disjointes. Des résultats expérimentaux montrent que MindTheGaps améliore de manière significative les performances de l'algorithme de résolution classique.

En général, un backtracking chronologique est utilisé pour gérer les sous-problèmes générés par l'étape de décomposition. Il existe pourtant d'autres méthodes plus sophistiquées comme par exemple le backtracking dynamique [64]. MindTheGaps est compatible avec n'importe quelle technique de backtracking.

Une approche similaire avait déjà été suggérée par Hansen [48, 49] pour la méthode de Newton par intervalles. L'algorithme de recherche exploite les trous identifiés lors de l'étape de résolution par l'algorithme de Gauss-Seidel. Cette approche a également été utilisée par Ratz [103] pour traiter des problèmes d'optimisation. Trois différentes stratégies de décomposition ont été explorées par Ratz :

- Utiliser seulement le trou le plus grand pour décomposer le domaine et générer 2 sous-problèmes [48].
- Utiliser au plus k trous dans le même domaine pour décomposer le domaine et générer $k+1$ sous-problèmes [103].
- Utiliser au plus 3 trous dans 3 domaines différents, combiner les sous-domaines pour générer au plus 8 sous-problèmes [49].

En fait, dans ce chapitre nous généralisons l'approche de Hansen pour tous les algorithmes de filtrage par consistances : Hull-consistance, Box-consistance and kB-consistances. En particulier, nous montrons comment ces techniques de filtrage identifient des trous dans les domaines. Nous montrons également que cette approche est efficace pour résoudre des CSPs numériques, c'est-à-dire pour trouver toutes les solutions isolées ou tous les espaces de solutions isolés.

Hyvönen [53] a également utilisés les trous mais pour calculer une consistance plus forte. Il a proposé un algorithme pour calculer la consistance d'union en combinant des ensembles d'intervalles, mais sa méthode est limitée par son caractère fortement exponentiel. MindTheGaps utilise les trous uniquement pour guider la recherche, ce qui limite le nombre de trous

générés. Pour limiter le coût de la gestion des unions d'intervalles, les trous identifiés par les contraintes trigonométriques ne seront pas prises en compte. L'identification des trous sera restreinte aux puissances et aux produits de variables, dont la projection ne produit qu'un seul trou. Nous y reviendrons dans la section 8.3.

Dans ce chapitre, nous utiliserons des unions d'intervalles, notées $\boldsymbol{u} = \bigcup \boldsymbol{u}_{(j)}$, où les sous-intervalles $\boldsymbol{u}_{(j)}$ sont disjoints et triés par borne inférieure croissante, *i.e.* $\overline{u}_{(j)} < \underline{u}_{(j+1)}$. Le nombre de sous-intervalles de $\boldsymbol{u}$ est dénoté par $|\boldsymbol{u}|$. La borne inférieure (resp. supérieure) de $\boldsymbol{u}$ est notée $\underline{\boldsymbol{u}}$ (resp. $\overline{\boldsymbol{u}}$). $\mathbb{U}$ dénote l'ensemble des unions d'intervalles et $\cap_{\mathbb{U}}$ est l'opérateur d'intersection sur $\mathbb{U}$, tel que $\mathbf{u} \cap_{\mathbb{U}} \mathbf{v} = \{x \in \mathbb{R} : x \in \mathbf{u} \wedge x \in \mathbf{v}\}$. Enfin, les symboles $\boldsymbol{U}, \boldsymbol{V}$ sont utilisés pour représenter des vecteurs d'unions d'intervalles.

Notations Dans la suite, on utilisera les notations[1] suivantes :

$\boldsymbol{u} = \bigcup \boldsymbol{u}_{(j)}$	une union d'intervalles
$\lvert\boldsymbol{u}\rvert$	le nombre de sous-intervalles qui composent $\boldsymbol{u}$
$\underline{\boldsymbol{u}}$	la borne inférieure de $\boldsymbol{u}$
$\overline{\boldsymbol{u}}$	la borne supérieure de $\boldsymbol{u}$
$\mathbb{U}$	l'ensemble des unions d'intervalles sur à bornes dans $\mathbb{F}$
$\boldsymbol{U}, \boldsymbol{V}$	des vecteurs d'unions d'intervalles
$\cap_{\mathbb{U}}$	l'opérateur d'intersection sur $\mathbb{U}$
$\cap_{\mathbb{I}}$	l'opérateur d'intersection sur $\mathbb{I}$

Ce chapitre est organisé de la manière suivante : La section 8.2 donne une vue d'ensemble de l'algorithme MindTheGaps. La section 8.3 décrit les extensions des algorithmes de filtrage par Hull-consistance et Box-consistance qui collectent les trous. La section 8.4 montrent quelques résultats expérimentaux sur des problèmes reconnus. Enfin, la section 8.5 propose différentes extensions de la méthode pour les consistances partielles.

8.2 MindTheGaps : Schéma général

Les techniques classiques pour résoudre les CSPs numériques sont basées sur un algorithme de type Branch&Prune (voir figure 3.1 page 32). Cet algorithme alterne réduction des domaines et décomposition de domaines jusqu'à ce qu'une précision donnée sur les boîtes ω_{sol} soit atteinte. En général, la bisection est utilisée et les intervalles sont coupés en leur milieux. Différentes stratégies de sélection de domaines peuvent être combinées à la bisection, dont RR, LF ou MS, déjà mentionnées dans la section 3.4.1 page 32.

Contrairement aux techniques classiques de sélection de domaines, MindTheGaps (voir figure 8.1) exploite les trous produits par les algorithmes de filtrage par consistance.

La fonction Prune* (ligne 5) collecte les trous générés pendant l'étape de filtrage. Les trous identifiés sont stockés dans $\boldsymbol{U}$, qui est un vecteur d'unions d'intervalles $(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$, tel que :

$$\boldsymbol{u}_i = \bigcup \boldsymbol{u}_{i(j)},$$

, où $\boldsymbol{u}_{i(j)} = [\underline{u}_{i(j)}, \overline{u}_{i(j)}]$ est le j-ème sous-domaine de x_i.

[1] Un récapitulatifs des notations utilisées dans ce manuscrits est présenté à la page x

Tant que $w(\boldsymbol{X})$ est plus grand qu'un ω_{sol} donné, MindTheGaps décompose en premier les domaines qui contiennent au moins un trou (ligne 11). Différentes heuristiques pour sélectionner le domaine à découper ou pour choisir les points de coupe ont été explorées. Nous y reviendrons dans la section 8.4. La découpe des domaine proprement dite est mise en œuvre par la fonction GapSplit, qui élimine un ou plusieurs trou du domaine sélectionné, générant ainsi plusieurs sous-problèmes qui sont stockés dans l'ensemble Q.

```
    MindTheGaps(in :X_0, C, ω_sol out : S)
    %% X_0 = (x_1, ..., x_n)
1 :  Q ← {X_0}
2 :  S ← ∅
3 :  while Q ≠ ∅ do
4 :    Extract X from Q
5 :    X ← Prune*(C, X, U)
6 :    if  X ≠ ∅  then
7 :      if  w(X) ≤ ω_sol  then
8 :        S ← S ∪ X
9 :      else
10 :       if ∃k s.t. |u_k| > 1 then
             % Gap Splitting
11 :         Q ← Q∪ GapSplit(U)
12 :       else
             % Standard splitting process
13 :         Q ← Q∪ Split(X)
14 :       endif
15 :     endif
16 :   endif
17 :  endwhile
18 :  return S
```

FIG. 8.1 – Schéma général de MindTheGaps.

MindTheGaps découpe d'abord les domaines qui contiennent des trous. Si aucun trou n'a été identifié par le filtrage, un splitting classique est mis en œuvre en combinaison avec l'une des stratégies de sélection classique (RR, LF, ou MS).

La section suivante présente les algorithmes de filtrage par consistance étendus de manière à collecter les trous qu'ils produisent.

8.3 Consistances locales et trous

La plupart des solveurs de contraintes (par exemple, IlogSolver [54], Numerica [119], Realpaver [46]) sont basés sur des consistances locales (Hull-consistance [79, 13], Box-consistance [119, 14]). Les algorithmes de filtrage associés réduisent les domaines des variables en éliminant des valeurs pour lesquelles certaines contraintes ne sont pas vérifiées (inconsistance). Cette réduction est calculée par des opérateurs de narrowing, qui sont des fonctions correctes, monotones et contractantes (voir définition 3.5.1 page 34). Ces réductions sont propagées en

utilisant l'algorithme standard de propagation, dérivé d'AC3 [84] (voir algorithme Prune, figure 3.2 page 34).

La Hull-consistance et la Box-consistance sont toutes les deux basées sur l'algorithme Prune, mais avec un opérateur de réduction spécifique. Autrement dit, la Hull-consistance et la Box-consistance implémente chacune leur propre version de la fonction Narrow.

Dans cette section, nous décrivons une façon d'étendre ces opérateurs afin qu'ils collectent les trous identifiés par la technique de consistance à laquelle ils sont associés. Ces versions étendues, nommées $\mathsf{Narrow}^\star(c_i, \boldsymbol{X}, \mathbf{U})$, stockent les trous dans le vecteur d'union d'intervalles $\mathbf{U} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$. À partir de ces opérateurs étendus, on définit naturellement des algorithmes de filtrage par Hull-consistance et par Box-consistance qui collectent les trous qu'ils identifient.

La section suivante rappelle quelques notions de base sur l'artihmétique des intervalles qui sont nécessaires à la définition de ces algorithmes.

8.3.1 Extensions aux intervalles et fonctions de projection

Dans la section 3.3 page 27, on avait défini les extensions aux intervalles des fonctions ainsi que les fonctions de projection (Déf. 3.3.2 page 29 et 3.5.2 page 34). Le point clé est que ces fonctions de projection peuvent générer des unions d'intervalles comme on l'avait montré dans la figure 3.3 page 35.

Les opérateurs de narrowing de la Hull-consistance utilisent ces fonctions de projection pour réduire le domaine des vairables. De manière informelle, $\pi_k(c \cap \boldsymbol{X})$ (Déf. 3.5.2 page 34) dénote la projection sur le domaine de la variable x_k des solutions de c lorsque les valeurs des variables sont restreinte à la boîte $\boldsymbol{X}$. $\pi_k(c \cap \boldsymbol{X})$ peut être approximée par le plus petit intervalle englobant, noté $\square_{\mathbb{I}}(\pi_k(c \cap \boldsymbol{X}))$, ou par la plus petite union d'intervalles, notée $\square_{\mathbb{U}}(\pi_k(c \cap \boldsymbol{X}))$.

Exemple 8.3.1 *Considérons la contrainte $c : y = x^2$, avec $x \in [-2, 4]$ et $y \in [1, 16]$ et soit $\boldsymbol{X} = [-2, 4] \times [1, 16]$. La figure 3.3 page 35 montre que l'approximation par intervalle de $\pi(c_\cap \boldsymbol{X})$ est $\square_{\mathbb{I}}(\pi(c_\cap \boldsymbol{X})) = [-2, 4]$, alors que l'approximation par union d'intervalles de $\pi(c_\cap \boldsymbol{X})$ is $\square_{\mathbb{U}}(\pi(c_\cap \boldsymbol{X})) = [-2, -1] \cup [1, 4]$.*

Les projections des fonctions trigonométriques, comme sinus ou cosinus, peuvent engendrer un nombre très important de trous. Pour limiter les coût de la gestion des unions d'intervalles, nous limitons l'identification des trous aux termes produits et puissances, qui ne peuvent produire au plus qu'un trou. Notons que plusieurs trous peuvent être produit dans le même domaine en intersectant les projections de différentes contraintes. La forme syntaxique des contraintes influence de manière significative sur l'identification de trous. Nous y reviendrons dans la section 8.4.

La section suivante présente l'algorithme de filtrage par Hull-consistance étendu afin de collecter les trous.

8.3.2 Hull-consistance et trous

Comme on l'avait déjà enoncée dans la section 3.5.1 page 33, la Hull-consistance établit une propriété de consistance locale sur les bornes de domaines. Une contrainte c est dite Hull-consistante si pour toute variable x_i de $\mathcal{X}(c)$, il existe une valeur dans les domaines des autres variables qui satisfait c lorsque x_i est instanciée à $\underline{x}_i$ ou à $\overline{x}_i$. Rappelons la définition ici :

Définition 8.3.1 (Hull-consistance [79, 15]) *Soit c une contrainte n-aire. c est Hull-consistante sur la boîte $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ ssi $\forall x_k \in \mathcal{X}(c), \boldsymbol{x}_k = \square_{\mathbb{I}}(\pi_k(c \cap \boldsymbol{X}))$. Un CSP est Hull-consistant ssi toutes ses contraintes sont Hull-consistantes.*

L'implantation basique de la Hull-consistance, nommée 2B-consistance [79, 80], décompose le système de contraintes en contraintes primitives pour lesquelles il est plus facile de calculer les fonctions de projection. L'implantation la plus efficace de la Hull-consistance est HC4 [13], basée sur l'opérateur HC4Revise. Cette implantation ne nécessite pas l'ajout de contraintes supplémentaire ; toutes les projections sont calculées an parcourant une représentation arborescente des contraintes, de bas en haut puis inversement (voir section 3.5.1 page 33 pour plus de détails).

Détaillons à présent HCNarrow, l'opérateur de réduction associé à la Hull-consistance, c'est-à-dire, HCNarrow correspond à la fonction Narrow dans l'algorithme générique de filtrage par propagation (voir figure 3.2 page 34).

```
    HCNarrow(in :c, X) : Interval vector
    % X = (x_1, ..., x_n)
1 : foreach x_k ∈ 𝒳(c) do
2 :   x_k ← □_𝕀(x_k ∩_𝕌 □_𝕌(π_k(c∩X)))
    % Les éventuels trous sont perdus
3 :   if x_k = ∅ then
4 :     return ∅
5 :   endif
6 : endfor
7 : return X
```

FIG. 8.2 – HCNarrow : l'opérateur de réduction de la Hull-consistance

Basiquement, HCNarrow (figure 8.2) réduit le domaine $\boldsymbol{X}$ par rapport à la contrainte c en appliquant l'opérateur de réduction à chaque variable $x_k \in \mathcal{X}(c_c)$. Cet opérateur de réduction réduit les bornes du domaine $\boldsymbol{x}_k$ en calculant une enveloppe par union d'intervalles de la fonction de projection $\pi_k(c)$, notée $\square_{\mathbb{U}}(\pi(c_{\cap}\boldsymbol{X})$.

Cette évaluation par union est alors intersectée avec $\boldsymbol{x}_k$ (ligne 2), et les trous éventuels sont perdus durant cette opération. En fait, les trous ne sont utilisés que pour calculer une évaluation plus fine du filtrage de $\boldsymbol{x}_k$.

Cependant, ces trous peuvent être récupérés en rempaçant l'opérateur d'intersection sur les intervalles ($\cap_{\mathbb{I}}$) par son correspondant sur les unions d'intervalles ($\cap_{\mathbb{U}}$). L'inconvénient est que le calcul d'intersection sur les unions d'intervalles coûte plus cher que celui sur les intervalles. Par conséquent, cela rajouterait un coût qui au final peut peser dans les performances du filtrage.

Or, il n'est pas nécessaire de calculer cette intersection d'unions d'intervalles à chaque projection. Les opérateurs de réduction étant contractants ($\Phi(\Omega) \subseteq \Omega$), la dernière projection opérée par l'algorithme de propagation fournira la plus petite union d'intervalles (au sens de l'inclusion d'ensembles), c'est-à-dire le plus grand trou. Autrement dit, il suffit de conserver pendant la propagation un ensemble S de couples contraintes/variables de la forme (c, x_k) pour lesquels des trous ont été identifiés. Il suffira alors de recalculer une approximation des projections $\pi_k(c)$ par union d'intervalles à la fin de la propagation.

Plus précisemment, l'algorithme HCNarrow* (voir figure 8.3) vérifie si l'évaluation de $\pi_k(c)$ produit un trou dans le domaine $\boldsymbol{x}_k$ (lignes 6-10). Dans ce cas, le couple (c, x) est ajouté à S, sinon (c, x) est retiré de S pour traiter le cas où un trou avait précédemment été trouvé dans le domaine de x mais que celui-ci ait été repoussé hors du domaine durant la phase de propagation.

```
    HCNarrow*(in :c, X, in-out : S) : Interval vector
    % X = (x_1, ..., x_n)
1 : foreach x_k ∈ V_c do
2 :   u ← x_k ∩_U □_U(π_k(c ∩ X)))
3 :   if u = ∅ then
4 :     return ∅
5 :   else
        % Mind the gap
6 :     if |u| > 1 then
7 :       S ← S ∪ (c, x_k)
8 :     else
9 :       S ← S \ (c, x_k)
10 :    endif
11 :    x_k ← □_I(u)
12 :  endif
13 : endfor
14 : return X
```

FIG. 8.3 – HCNarrow* : extension de HCNarrow qui collecte les trous

Soit HCPrune$^+$($\boldsymbol{C}$,$\boldsymbol{X}$,S), l'algorithme Prune (voir figure 3.2 page 34), dans lequel l'appel à l'opérateur de réduction a été remplacé par HCNarrow*(c_i,$\boldsymbol{X}$,S). HCPrune$^+$ établit la Hull-consistance sur la boîte $\boldsymbol{X}$ et collecte dans l'ensemble S les paires contraintes/variables pour lesquelles des trous ont été identifiés. Enfin, HCPrune* récupère ces trous (lignes 4-6) en intersectant $\boldsymbol{u}_k$ avec l'évaluation par union d'intervalles de la projection $\pi_k(c)$ (voir figure 8.4).

```
    HCPrune*(in :C, in-out : X, U)
    % X = (x_1, ..., x_n)
    % U = (u_1, ..., u_n)
1 : S ← ∅
2 : HCPrune+(C,X,S)
3 : if X ≠ ∅ then
      % Collect the gaps
4 :   foreach (c, x_k) ∈ S do
5 :     u_k ← u_k ∩_U □_U(π_c^{x_k}(X))
6 :   endfor
7 : endif
```

FIG. 8.4 – HCPrune* enforces Hull-consistency and collects the gaps

La section suivante présente l'extension de l'algorithme de fitrage par Box-consistance qui collecte les trous.

8.3.3 Box-consistance et trous

Comme on l'avait déjà enoncée dans la section 3.5.1 page 33, la Box-consistance [14, 119] est une approximation plus grossière de l'arc-consistance que la Hull-consistance. Informellement, une contrainte c est Box-consistante si pour toutes les variables x_i de $\mathbb{V}_c$, les bornes de $\boldsymbol{x}_i$ satisfont la contraintes unaire obtenu en remplaçant chaque occurrence d'une variable x_j autre aue x_i par l'intervalle constant $\boldsymbol{x}_j$. Rappelons la définition de la Box-consistance :

Définition 8.3.2 (Box-consistance [14, 119]) *Soit c une contrainte n-aire. c est Box-consistante pour la boîte $\boldsymbol{X}$ ssi $\forall x_i$ les deux propriétés suivantes sont vérifiées :*

1. $\boldsymbol{c}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, [\underline{x}_i, \underline{x}_i^+), \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)$
2. $\boldsymbol{c}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, (\overline{x}_i^-, \overline{x}_i], \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_n)$

Un CSP est Box-consistant si toutes ses contraintes sont Box-consistantes.

La Box-consistance génère des ensemble de fonctions univariées qui peuvent être résolue par des méthodes numériques comme la méthode de Newton [49]. Le filtrage consiste à rechercher les quasi-zéros les plus à droite et les plus à gauche de ces fonctions univariées. La fonction BCNarrow (voir section 3.5 page 33), réduit le domaine de toutes les variables d'une contrainte c jusqu'à ce que c soit Box-consistante. Le filtrage du domaine d'une variable de c consiste à réduire les borne de son domaine au quasi-zéros extrème de la fonction univariée $\boldsymbol{f}_{\boldsymbol{x}_k}$. Cette réduction est opérée par la fonction LeftNarrow (see figure 3.6 page 37) pour la borne inférieure et par LeftNarrow (see figure 3.6 page 37) pour la borne supérieure.

Ces fonctions sont basées sur MonoNewton qui réduit le domaine de la variable $\boldsymbol{x}$ par rapport à la contrainte c, en utilisant l'algorithme classique de Newton monovarié par intervalles. Tant que la réduction de $\boldsymbol{x}$ est inférieure à ϵ, une dichotomie est appliquée pour garantir que $\boldsymbol{x}$ est bien un quasi-zéro. LeftNarrow* (voir figure 8.5) collecte les trous identifiés par l'opérateur de réduction de la Box-consistance.

Le point clé est que l'appel à MonoNewton (ligne 5) produit des trous de deux manières différentes :

1. Si la borne gauche de l'intervalle courant $\boldsymbol{x}$ est réduit par la méthode de Newton, alors l'intervalle éliminé $[\overline{x}', \overline{x}]$ ne satisfait pas la contrainte c. Or cet intervalle est strictement inclus dans le domaine initial de la variable x et consistue un trou.
2. Par la méthode de Newton, MonoNewton, elle-même.

Pour expliquer comment MonoNewton peut produire des trous, rappelons la définition de la méthode de Newton par intervalles :

$$\begin{array}{rcl} \boldsymbol{x}^{(0)} & = & \boldsymbol{x} \\ \boldsymbol{x}^{(n+1)} & = & N(\boldsymbol{f}, \boldsymbol{f}', \boldsymbol{x}^{(n)}), \\ \text{where } N(\boldsymbol{f}_{\boldsymbol{x}}, \boldsymbol{f}'_{\boldsymbol{x}}, \boldsymbol{x}^{(n)}) & = & \boldsymbol{x}^{(n)} \cap (m(\boldsymbol{x}^{(n)}) - \frac{\boldsymbol{f}_{\boldsymbol{x}}(m(\boldsymbol{x}^{(n)}))}{\boldsymbol{f}'_{\boldsymbol{x}}(\boldsymbol{x}^{(n)})}) \end{array}$$

La fonction MonoNewton calcule une fermeture de $N(\boldsymbol{f}_{\boldsymbol{x}}, \boldsymbol{f}'_{\boldsymbol{x}}, \boldsymbol{x})$ et retourne l'intervalle résultant. Or, l'évaluation de la division $\frac{\boldsymbol{f}(m(\boldsymbol{x}^{(n)}))}{\boldsymbol{f}'(\boldsymbol{x}^{(n)})}$ en utilisant l'arithmétique étendue [49, **?**] peut produire un trou, comme le montre l'exemple suivant :

```
    LeftNarrow*(in : f, f', x, in-out : S, u) : Interval
1 :  r ← x̄
2 :  if 0 ∉ f(x) then
3 :    return ∅
4 :  endif
5 :  u' ← MonoNewton*(f, f', x)
     % Mind the gap
6 :  u ← u ∩_𝕌 (u' ∪ [r, +∞))
7 :  x ← □_𝕀(u)
8 :  if 0 ∈ f([x̲, x̲⁺]) then
9 :    return [x̲, r]
10 : else
11 :   l ← LeftNarrow*(f, f', [x̲, m(x)], S)
12 :   if l = ∅ then
13 :     l ← LeftNarrow*(f, f', [m(x), x̄], S)
14 :   endif
15 :   return [l̲, r]
16 : endif
17 : return x
```

FIG. 8.5 – Opérateur de réduction de la Box-consistance étendu pour collecter les trous

Exemple 8.3.2 *Soit* $f(x,y) = x^2 - y$ *avec* $x \in [-4,4]$ *et* $y \in [1,16]$. *La fonction intervalle* $\boldsymbol{f_x}$ *et sa dérivée derivative* $\boldsymbol{f'_x}$ *sont définies par* $\boldsymbol{f_x}(x) = x^2 - [1,16]$ *et* $\boldsymbol{f'_x}(x) = 2x$. *Donc,*

$$\begin{aligned}
\boldsymbol{x}^{(0)} &= [-4,4] \\
\boldsymbol{x}^{(1)} &= [-4,4] \cap (0 \ominus ((0^2 \ominus [1,16]) \oslash (2 \otimes [-4,4]))) \\
&= [-4,4] \cap ([1,16] \oslash [-8,8]) \\
&= [-4,-1/8] \cup [1/8,4]
\end{aligned}$$

Par conséquent, $N(\boldsymbol{f_x}, \boldsymbol{f'_x}, \boldsymbol{x})$ n'est pas en règle général un simple intervalle, mais peut-être une union d'intervalles. Notons MonoNewton*, la fonction qui retourne cette union d'intervalles.

On définit alors LeftNarrow* (voir figure 8.5), qui étend l'opérateur classique de réduction de la Box-consistance de manière à collecter les trous. MonoNewton* récupère les trous produits par la méthode de Newton monovairé étendue (ligne 5). Enfin, les trous produits par la réduction de la borne gauche sont récupérés (ligne 6).

La section suivante présente quelques résultats expérimentaux qui montrent une amélioration significative du temps de résolution par rapport aux techniques de recherche classique.

8.4 Résultats expérimentaux

Cette section présente des résultats expérimentaux de MindTheGaps sur une variété de problèmes classiques : deux benchmarks classiques issus de l'arithmétique des intervalles (*i1*,*i4*), une application de cinématique directe de robot (*kin1*), un problème de mécanique céleste à 5 corps (*nbody5*), quelques applications de modélisation économique (*eco7*, *eco8*, *eco9* and *eco10*), des problèmes de satisfaction de contraintes de distance (*ponts*, *ext-penta* et

des instances particulières), et un système issu de la liste de problèmes de Posso (*caprasse*). Ces problèmes sont détaillés dans la littérature, notamment dans [119], pour *i1, i4, kin1* and *ecoN*, dans [68] pour *nbody5*, dans [62] pour *ponts* et dans [116] pour *caprasse*. Le problème du pentagone étendu, *ext-penta* a quant à lui été défini dans la section 7.4.1.

La section suivante présente trois catégories d'heuristiques pour personnaliser MindTheGaps :

- Déterminer quels trous sont les moins pertinents et peuvent être éliminés (par exemple, des trous trop petit).
- Sélectionner des domaines contenant des trous.
- Découper le(s) domaine(s) seléctionné(s) en uilisant les trous.

Ensuite, dans la section 8.4.2 les résultats de MindTheGaps sur les benchmarks seront décrits pour la Hull et pour la Box. Dans la section 8.4.3, on verra comment la forme syntaxique influence l'identification des trous et son impact sur les performances de MindTheGaps.

8.4.1 MindTheGaps : heuristiques

Cette section présente trois cétégories de stratégies pour personnaliser MindTheGaps. Ces heuristiques ont été explorées dans le but de répondre au 3 questions suivantes :

1. Pertinence des trous : Quels trous sont insignifiants et devraient être ignorés ?
2. Sélection de domaines : Parmi les domaines dans lesquels des trous ont été identifiés, lesquels présentent les choix de coupe les plus intéressants ?
3. Gap splitting : Quel(s) trou(s) doit-on éliminer pour décomposer le ou les domaines seléctionnés ?

- *Stratégies de validation des trous* : Supposons qu'un trou a été identifié durant le processus de filtrage dans le domaine $\boldsymbol{x} = [a, d]$, tel que $\boldsymbol{u} = [a, b] \cup [c, d]$. Deux différentes stratégies ont été explorée pour valider le trou (b, c), en fonction de sa position à l'intérieur du domaine ou de sa proportion.
  - Hansen [49] : Ne considérer (b, c) que si $\min\{d - b, c - a\} \geq 0.25w(\boldsymbol{x})$. Cette stratégie élimine les trous qui sont entièrement inclus dans un des deux quart extrème du domaine.
  - Large-Gaps : Ne considérer (b, c) que si $c - b \geq 0.1w(\boldsymbol{x})$. Cette stratégie élimine des trous dont la taille relative est inférieure à 10% de la taille du domaine.

  Notons que ces deux stratégies peuvent être combinées. Par défaut, tous les trous sont conservés (AllGaps).
- *Stratégies de sélection de domaines* : Différentes heuristiques ont été explorées, pour la plupart basées sur la taille des trous identifiés :
  - LW (Largest Width) /SW (Smallest Width) : Le domaine sélectionné contient le trou dont la taille est la plus grande (resp. petite) [49].
  - LRW (Largest Relative Width) /SRW (Smallest Relative Width) : Le domaine sélectionné maximise (resp. minimise) le rapport entre la taille de l'un de ses trous et sa taille.
  - LTW (Largest Total Width) /STW (Smallest Total Width) : Le domaine sélectionné maximise (resp. minimise) la somme des tailles de ses trous.
- *Stratégies de décopomposition basée sur les trous* : Trois stratégies de décomposition ont été explorées :
  - B1G (Bisect One Gap) : Utiliser un seul trou pour décomposer le domaine sélectionné et générer 2 sous-problèmes [48]. Le trou est déterminé par la stratégie de sélection

de domaine.
- BkG (Bisect k Gaps) : Utiliser au plus k trous dans le domaine sélectionné pour décomposer la boîte et générer $k+1$ sous-problèmes [103].
- M3G (Multisect 3 Gaps) : Utiliser au plus 3 trous dans 3 domaines différents et combiner les sous-domaines pour générer au plus 8 sous-problèmes [49]. Les trois domaines et les trous sont déterminés là encore par la stratégie de décomposition.

Dans la section suivante, les résultats expérimentaux sont présentés et analysés.

8.4.2 Analyse des résultats

Les différentes heuristiques mentionnées précedemment ont été explorées méthodiquement sur les différents benchmarks. Les tables présentées dans cette section se limitent aux résultats obtenus avec RR,LW et B1G. Les autres stratégies ont donné des résultats tout à fait similaires pour la plupart des benchs, sauf pour ceux que l'on détaillera dans le reste de cette section.

Tous les tests ont été réalisés sur la plate-forme Realpaver [46] version 0.3, sur un Pentium IV cadencé à 2.6Ghz sous Linux. MindTheGaps a été interfacé avec Realpaver. Les différentes stratégies ainsi que la collecte des trous ont été ajoutés aux algorithmes par défaut de Realpaver. Notons que l'algorithme de Newton multivarié a aussi été étendu afin de collecter les trous, selon la méthode proposée par Hansen [49]. L'algorithle de filtrage par Box-consistabce a été modifié afin de coller aux spécification de l'algorithme par défaut [14, 119], qui utilise le Newton monovarié. En effet, l'implantation de la Box sous Realpaver n'utilisait pas le Newton monovarié, mais se basait uniquement sur l'évaluation par intervalle pour calculer les quasi-zéros.

Les tables 8.1, **??**, 8.3, 8.4 montrent les résultats obtenus sur la série de problèmes présentés en introduction de cette section (p. 95). Tous les résultats présentés ont été obtenus avec RR,LW and B1G. Dans ces tableaux, la colonne t indique le temps de calcul ("-" désigne un temps de calcul supérieur à 1h), B est le nombre total de boîtes générées par le processus de splitting, et H est le nombre de fois qu'un trou a servi à décomposer le domaine. La colonne "ratio" présente le pourcentage de réduction en terme de temps CPU (t) et de nombre de branchements (B).

La table 8.1 montre les résultats obtenus pour des résolutions basés sur des filtrage par HC4 et HC4 combiné avec l'algorithme de Newton multivarié. Dans les deux cas, MindTheGaps améliore significativement le temps d'exécution et réduit le nombre de boîtes générées par le splitting. Par exemple, pour *eco7*, le temps d'exécution a été réduit d'un facteur 3.8 ou plus, selon le filtrage utilisé, et le nombre de branchements a également été réduit d'un facteur compris entre 3 et 4.

Même sur des problèmes où le nombre de branchements reste inchangé, comme par exemple pour *i4*, MindTheGaps améliore le temps de résolution. Ce problème met en évidence le rôle que joue l'élimination des trous dans le choix des points de coupe, mais également dans la réduction de l'espace de recherche.

D'autres stratégies que de choisir la variable dont le domaine contient le plus grand trou et couper en utilisant ce trou ne changent pas de manière significative les résultats. Cependant, les performances sont améliorées de manière significatives pour la Box ou la Box combinée avec la méthode de Newton multivarié lorsqu'on éliminent les trous qui se situent trop loin du centre de l'intervalle (critère de Hansen) ou ceux qui sont trop petits (voir table **??** et 8.3). Par exemple, le problème *eco8* a pu être résolu en moins d'une demi-heure alors que les autres stratégies requéraient plus d'une heure de temps de calcul. Ce succès est dû à la façon dont

	Filtering : HC4							Filtering : HC4+Newton						
	RR		MindTheGaps(RR)			Ratio		RR		MindTheGaps(RR)			Ratio	
	t(s)	B	t(s)	B	H	t	B	t(s)	B	t(s)	B	H	t	B
eco7	57.07	754885	14.74	231595	1	**-74**%	**-69**%	61.99	468799	12.74	107817	11	**-79**%	**-77**%
eco8	133.51	1614495	112.77	1360061	1	**-15**%	**-16**%	56.77	353155	40.54	246927	49	**-29**%	**-30**%
ponts	34.19	174915	33.12	171251	1043	**-3**%	**-2**	25.61	32643	16.80	21025	946	**-35**%	**-36**%
ponts0	0.30	1395	0.29	1395	0	-	-	0.06	71	0.07	71	0	-	-
ponts1	5.19	26523	4.54	22475	274	**-13**%	**-16**%	5.78	7465	3.61	4563	254	**-38**%	**-39**%
ponts2	23.63	123585	22.93	120993	779	**-3**%	**-2**%	18.77	24009	12.33	15567	670	**-35**%	**-35**%
pentagon	0.60	6131	0.59	5891	52	**-2**%	**-4**%	0.26	1655	0.23	1415	52	**-12**%	**-15**%
ext-penta	-	-	-	-	-	-	-	474.92	1006031	437.37	890943	11723	**-8**%	**-12**%
ext-penta0	0.34	873	0.11	423	51	**-68**%	**-52**%	0.45	873	0.17	423	51	**-62**%	**-52**%
ext-penta1	0.32	263	0.05	255	17	**-85**%	**-3**%	0.39	263	0.10	255	17	**-74**%	**-3**%
ext-penta2	0.77	2825	0.25	2047	9	**-68**%	**-28**%	1.23	2825	0.60	2047	9	**-51**%	**-28**%
nbody5	121.65	1756139	116.71	1645977	74882	**-4**%	**-7**%	74.70	841461	67.34	752079	84191	**-10**%	**-11**%
i1	29.60	515909	28.75	501677	82	**-3**%	**-3**%	49.46	340057	53.73	370449	38	8%	9%
i4	0.83	2047	0.77	2047	1023	**-8**%	-	1.19	2047	1.07	2047	1023	**-10**%	-
kin1	25.69	264685	19.99	203987	1	**-22**%	**-23**%	0.41	1447	0.30	1263	1	**-27**%	**-13**%
caprasse	0.79	6527	0.80	6527	0	1 %	-	0.65	2567	0.65	2567	0	-	- %

TAB. 8.1 – Résultats expérimentaux pour HC4 (à gauche) et HC4+Newton (à droite).

	Filtering : Box											
	RR		MindTheGaps(RR)			Ratio		MindTheGaps(RR)+Hansen			Ratio	
	t(s)	B	t(s)	B	H	t	B	t(s)	B	H	t	B
eco7	995.12	595505	-	-	-	-	-	276.91	192699	104	**-72**%	**-68**%
eco8	-	-	-	-	-	-		1372.13	847373	177	-	-
ponts	659.70	173331	644.60	170721	968	**-2**%	**-2**%	676.36	172515	1015	3%	**-1**%
ponts0	5.60	1481	4.90	1203	6	**-12**%	**-10**%	5.61	1481	0	-	- %
ponts1	105.47	25943	103.67	23335	122	**-2**%	**-10**%	99.83	22911	255	**-5**%	**-12**%
ponts2	461.20	122865	440.99	118123	656	**-5**%	**-4**%	459.61	121017	538	**-1**%	**-2**%
pentagon	11.27	6283	11.02	6275	173	**-3**%	- %	10.99	6033	51	**-3**%	**-4**%
ext-penta	-	-	-	-	-	-	-	-	-	-	-	-
ext-penta0	5.80	873	2.20	1771	666	**-62**%	102.86%	1.84	423	51	**-68**%	**-52**%
ext-penta1	6.65	263	1.09	873	306	**-84**%	232%	0.81	255	17	**-88**%	**-3**%
ext-penta2	11.36	2825	12.05	17865	7037	6%	533%	2.12	2047	9	**-82**%	**-28**%
nbody5	601.57	878023	567.60	792987	79132	**-6**%	**-10**%	563.18	783929	75018	**-7**%	**-11**%
i1	353.70	484511	369.60	502035	7614	5%	4%	353.67	482565	39	-	-
i4	5.51	2047	6.07	2047	1023	10%	-	6.05	2047	1023	10%	-
kin1	235.74	132547	-	-	-	-	- %	217.18	120309	56	**-8**%	**-10**%
caprasse	10.55	2023	10.50	1991	48	- %	**-2**%	10.52	1991	48	- %	**-2**%

TAB. 8.2 – Résultat expérimentaux pour Box (à gauche) et avec le critère de Hansen (à droite)

	Filtering : Box+Newton											
	RR		MindTheGaps(RR)			Ratio		MindTheGaps(RR)+Hansen			Ratio	
	t(s)	B	t(s)	B	H	t	B	t(s)	B	H	t	B
eco7	797.72	429263	207.32	109267	685	**-74**%	**-75**%	196.39	102487	145	**-76**%	**-76**
eco8	-	-	-	-	-	-	-	516.92	224513	371	-	-
ponts	163.31	31735	180.84	35475	1098	11%	12%	133.51	25829	1014	**-19**%	**-19**%
ponts0	0.47	71	0.47	71	0	-	-	0.47	71	0	-	- %
ponts1	33.78	7363	27.56	4981	261	**-18**%	**-32**%	27.39	4981	261	**-19**%	**-32**%
ponts2	117.27	23417	110.37	21881	646	**-6**%	**-7**%	96.50	18675	563	**-18**%	**-21**%
pentagon	3.76	1639	0.27	1399	62	**-93**%	**-15**%	3.45	1399	62	**-9**%	**-15**%
ext-penta	-	-	-	-	-	-	-	-	-	-	-	-
ext-penta0	6.43	873	1.58	707	133	**-75**%	**-19**%	2.02	423	51	**-69**%	**-52**%
ext-penta1	7.18	263	1.12	747	209	**-84.40**%	184%	0.92	303	41	**-88**%	15%
ext-penta2	12.55	2825	7.28	8721	3167	**-42**%	208.70 %	2.57	2047	9	**-80**%	**-28** %
nbody5	811.88	968213	770.64	881897	108255	**-5**%	**-9**%	766.25	876099	106005	**-6**%	**-10** %
i1	247.22	309681	244.86	302397	576	**-1**%	**-3**%	247.38	305797	52	-	**-1**%
i4	6.26	2047	6.84	2047	1023	9%	-	6.82	2047	1023	9%	-
kin1	2.78	791	1.87	641	72	**-33**%	**-19**%	1.86	629	66	**-33**%	**-20**%
caprasse	10.54	1495	10.52	1463	48	-	**-2**%	10.54	1463	48	-	**-2**%

TAB. 8.3 – Résultat expérimentaux pour Box+Newton (à gauche) et avec le critère de Hansen (à droite)

les trous sont générés par la Box-consistance. Le filtrage par Box-consistance tente d'éliminer des intervalles de plus en plus petit autour des bornes du domaine. Le résultat est que ce filtrage produit beaucoup de petits trous près des bornes des domaines. Ce comportement est mis en évidence sur *ext-penta2*, où le nombre de trous utilisés passe de 7037 à 9 (voir **??**). On peut remarquer la même chose quand la Box est combinée avec Newton, donc le critère d'Hansen tend à lisser le comportement de MindTheGaps combinée avec la Box.

Quelle que soit la stratégie, MindTheGaps améliore le temps d'éxécution comparée au classique Branch&Prune combiné avec le Round Robin. Mais, MindTheGaps a plus d'avantages que d'améliorer les performances de la recherche de solutions. Par exemple, sur le problème bien connu nommé *combustion*, MindTheGaps réussit à trouver les 4 solutions alors que le *Branch&Prune* classique ne génère que 2 boîtes englobant chacunes 2 solutions[2]. L'utilisation des trous permet donc dans certains cas d'isoler des solutions disjointes plus facilement qu'avec une stratégie de recherche standard.

La section suivante présente quelques résultats sur l'impact de la forme syntaxique des contraintes sur les performances de MindTheGaps.

8.4.3 Évaluation de contraintes et trous

Des règles de factorisation ont été conçues pour les polynômes univariés ou multivariés [25]. Ces outils symboliques tentent de réduire l'effet négatif des calculs par intervalles. En effet, l'arithmétique des intervalles a tendance à surestimer les résultats. En général, l'évaluation de contraintes polynômiales est plus fine sous forme factorisée. A priori, les trous produits sous cette forme sont également plus pertinents. Il est également possible que sous cette forme factorisée, certains trous puissent être découverts alors qu'ils le ne l'auraient pas été sous forme développée :

[2]Ces résultats ont été obtenus avec la précision par défaut de Realpaver (1.0e-8). En fixant la précision à 1.0e-12, Realpaver trouve toutes les solutions avec une stratégie de recherche standard.

	Filtering : HC4+Newton						
	RR		MindTheGaps(RR)			Ratio	
	t(s)	B	t(s)	B	H	t	B
eco6	1.04	12087	0.56	6383	3	**-46.15**%	**-47.19**%
eco6H	0.69	9301	**0.29**	3729	1	**-57.97**%	**-59.90**%
eco7	61.99	468799	**12.74**	107817	11	**-79.44**%	**-77.00**%
eco7H	49.78	412957	8.42	82143	4	**-83**%	**-80**%
eco8	56.77	353155	40.43	246927	49	**-28.78**%	**-30.07**%
eco8H	30.93	216955	**24.17**	164733	4	**-21.85**%	**-24.07**%
eco9	636.75	2931479	641.75	2934801	1720	.78%	.11%
eco9H	301.20	1541855	**233.67**	1303655	103	**-22.42**%	**-15.44**%
eco10	7569.05	25751025	7381.65	24939453	17949	**-2.47**%	**-3.15**%
eco10H	554.72	2620443	**475.00**	2156345	808	**-14.37**%	**-17.71**%

TAB. 8.4 – Résultats expérimentaux pour *ecoN* et les formes de Horner corresponantes *ecoNH*.

Exemple 8.4.1 *Considérons la contrainte* $c : x^2 + x * y = 1/2$ *et sa forme factorisée* c' : $x(x+y) = 1/2$, *avec* $\boldsymbol{x} = \mathbf{y} = [-1, 1]$. $\square_{\mathbb{U}}(\pi(c' \cap \boldsymbol{X})) = [-1, 1]$ *alors que* $\square_{\mathbb{U}}(\pi(c' \cap \boldsymbol{X}))) = [-1, 0.25] \cup [0.25, 1]$.

Nous avons effectués quelques expérimentations sur les problèmes *ecoN* afin de comparer les performances de MindTheGaps sur les formes dévelopées (*ecoN*) et factorisées (*ecoNH*). Ces expérimentations montrent que la forme de Horner (factorisée) apportent une amélioration significative (d'un facteur entre 2 et 15) par rapport aux formes développées pour la bisection standard. Le nombre de branchements dans lesquels des trous sont utilisés (Hà est fortement réduit (par exemple d'un facteur 16 pour *eco9*). Pourtant, l'impact de MindTheGaps est renforcé tant sur le temps de calcul que sur le nombre de branchements. Ces résultats montrent que les trous générés par l'évaluation des contraintes factorisées sont plus pertinents.

La section suivante discute de l'extension de MindTheGaps aux consistances partielles.

8.5 Extension aux consistances partielles

Les kB-consistances ne sont pas des consistances strictement locale. Ces consistances tentent de réfuter une partie du domaine en prouvant qu'il n'existe pas de solutions dans cette partie. Pour ce faire, elles utilisent une consistance d'ordre inférieure, la $(k-1)-$*consistance*. Le point clé est qu'elles tentent de manière assez similaire à la Box-consistance de réduire les bornes des domaines en rejetant $[\underline{x}, s]$ ou $[s, \overline{x}]$ où $s \in [\underline{x}, \overline{x}]$.

Les consistances partielles sont donc basées sur la 2B-consistance. Par conséquent, elles permettent également d'identifier des trous dans les domaines.

Chaque fois que la kB-consistance essaie de réfuter un intervalle$\alpha \subset \boldsymbol{x}_i$, elle applique un filtrage par $(k-1)$B-consistency sur $P_{\boldsymbol{x}_i \leftarrow \alpha}$. Si on suppose que α n'est pas éliminé mais réduit à α', les trous peuvent être collectés de 2 manière différentes :

1. kB-trous : $\alpha \setminus \alpha'$ est un trou pour $\boldsymbol{x}_i$
2. $(k-1)$B-trous : Les trous identifiés par la $(k-1)$B-consistance à l'intérieur de α' sont également valides pour $\boldsymbol{x}_i$.

Des tests expérimentaux ont été réalisés pour la 3B-consistance, en différenciant les 2B-trous et les 3B-trous. Plus préciément, la table 8.5 montre les résultats obtenus en utilisant

problem	3B	3B+B1G				3B+B1G+hansen			
		2B- & 3B-gaps		3B-gaps		2B- & 3B-gaps		3B-gaps	
	t(s)	t(s)	H	t(s)	H	t(s)	H	t(s)	H
cyclohexane	13.10	11.27	8	**11.26**	8	13.28	3	13.23	3
d1	10.30	7.63	9	6.04	5	7.64	9	**6.06**	5
dhingra	8.17	**7.85**	1	7.87	1	7.80	1	7.86	1
ext-penta	87.76	32.34	92	**32.15**	92	32.30	92	32.23	92
ext-penta0	4.36	2.70	31	2.68	31	**2.67**	31	2.70	31
ext-penta1	6.66	**2.85**	3	2.86	3	2.89	3	2.89	3
ext-penta2	21.90	11.09	31	11.08	31	11.23	31	**11.03**	31
i4	38.54	33.47	1023	**33.34**	1023	33.74	1023	33.39	1023
minass	8.27	**7.21**	4	7.25	4	7.26	4	**7.21**	4
octohedron0	20.12	16.77	2	16.79	2	16.85	2	**16.76**	2
octohedron1	20.10	16.84	2	**16.76**	2	16.81	2	**16.76**	2
octohedron2	61.53	40.35	1	**40.29**	1	40.37	1	40.44	1
pentagon	4.41	3.08	6	3.09	6	**3.07**	6	3.15	6
pentagone0	0.03	**0.01**	1	0.01	1	0.01	1	0.01	1
pentagone1	0.11	**0.04**	1	0.04	1	0.04	1	0.04	1
pentagone2	0.19	**0.11**	8	0.11	8	0.10	8	0.11	8
ponts0	0.11	0.11	0	0.12	0	0.11	0	0.10	0
ponts1	5.90	**5.23**	7	5.27	7	5.26	7	5.23	7
ponts2	31.86	**23.67**	33	25.54	37	23.70	33	25.72	37
pontsall	51.26	35.57	39	39.32	49	**35.47**	39	39.27	49
robot2D	7.08	6.46	4	6.47	4	6.48	4	**6.43**	4
seyfertfilter	290.62	**259.97**	7	260.28	7	260.37	7	260.48	7

TAB. 8.5 – Résultats expérimentaux de MindTheGaps pour la 3B-consistance

les 3B-trous uniquement (colonne 3B-gaps) et ceux obtenus en utilisant tous les trous (2B-gaps & 3B-gaps). MindTheGaps améliore les temps de résolution sur la plupart des problèmes. Toutefois, ces résultats n'indiquent pas de différence significative entre les 2B-trous et les 3B-trous. Le critère de Hansen ne change pas non plus significativement les temps de résolution.

Chapitre 9

Conclusion

Les contributions de cette thèse portent d'une part sur les contraintes globales en domaines continus et plus particulièrement pour les systèmes de contraintes de distance. D'autre part, nos travaux ont débouché sur deux techniques originales pour la résolution de CSPs à domaines continus : la première est spécifique aux contraintes de distance et la seconde est beaucoup plus générale.

Contraintes globales Nous avons tenté dans cette thèse de concevoir une contrainte globale pour les systèmes de contraintes de distance euclidienne. Ces systèmes sont définis par un ensemble de points et de distances entre certains couples de ces points. Le problème est de localiser ces points dans l'espace de manière à ce que les contraintes de distance soient respectées. Nous avons exploré deux différentes voies pour la conception d'une contrainte globale pour les systèmes de contraintes de distance : l'inférence de contraintes redondantes avec utilisation de filtrages standards, et la conception d'un algorithme de filtrage dédié.

1. **Ajout de contraintes redondantes** : Nous avons tenté d'inférer des contraintes additionnelles en se basant sur des propriétés géométriques élémentaires des systèmes de contraintes de distance.
   - *Fermeture du graphe de contraintes* : À partir du système initial, nous avons inféré un intervalle pour toutes les distances inconnues. Ces distances supplémentaires vérifient les inégalités triangulaires et garantissent que tout triangle du système est constructible. Un algorithme spécifique calcule cette fermeture avec une complexité en temps de l'ordre de $\Theta(n^3)$. La fermeture du graphe de contraintes de distance ajoute des contraintes au système initial qui permettent dans certains cas de détecter l'inconsistance du système plus rapidement.
   - *Introduction de barycentres* : L'algorithme de calcul de la fermeture du graphe de contraintes parcourt l'ensemble des triangles du système. Nous y avons incorporé une procédure de filtrage global pour les triangles. L'algorithme réalisé calcule la fermeture du graphe et filtre les domaines des points à la volée. Cette procédure de filtrage ajoute temporairement un point supplémentaire, l'isobarycentre du triangle, et évalue les distances entre ce point particulier et les sommets du triangle. L'ajout de ces contraintes et de ces barycentres au système initial permet un filtrage plus fort des domaines.

On peut généraliser cette approche à un ensemble de n points. Il est en effet possible de calculer une évaluation des distances entre chacun des points du système et leur isobarycentre :

$$n^2\delta_{Gk}^2 = n\sum \delta_{ik}^2 - \sum_{i<j} \delta_{ij}^2$$

Ces distances renferment une information globale sur le système, car elles sont définies à partir des distances entre tout couples de points. Il serait intéressant d'évaluer l'impact de ce système de contraintes supplémentaires sur le filtrage des domaines.

D'autres propriétés géométriques élémentaires pourraient être utilisées pour inférer des contraintes redondantes. Ces contraintes peuvent être éventuellement des contraintes angulaires ou d'incidence, d'orthogonalité ... Une autre extension possible pourrait être d'utiliser des prouveurs de théorèmes pour inférer des propriétés géométriques spécifiques au problème, puis d'utiliser ces propriétés comme des contraintes supplémentaires.

2. QuadDist **: Un filtrage global pour les contraintes de distance** :
Nous avons spécialisé la contrainte globale Quad [76, 89, 75], dédiée aux systèmes d'équations quadratiques, pour les contraintes de distance qui en sont un cas particulier. Cette méthode linéarise les termes quadratiques des équations (de la forme xy ou x^2), autrement dit Quad génère un certain nombre d'inéquations linéaires qui approximent l'espace des solutions. Ensuite, la méthode du simplexe est utilisée sur le système linéaire généré pour réduire les bornes des domaines.

   Nous avons introduit une nouvelle technique de linéarisation spécifiquement adaptée aux contraintes de distance. Cette nouvelle technique, nommée QuadDist, ne génère pas des linéarisations pour chaque terme des équations mais globalement pour chaque contrainte de distance. QuadDist définit donc une donc une approximation plus précise que Quad, ce qui augmente la vitesse de convergence de la méthode. D'autre part, contrairement à la Quad, notre linéarisation des contraintes ne nécessite pas l'ajout de variables supplémentaires ce qui réduit la taille des systèmes linéaires résolus par le simplexe. Les résultats expérimentaux montrent que QuadDist résout certains problèmes près de 300 fois plus vite que la Quad. En revanche, notre implantation de QuadDist ne garantit pas la correction des approximations linéaires calculées. L'adaptation des principes proposés dans [89] ne devrait toutefois pas poser de difficultés majeures.

   On peut envisager d'utiliser les linéarisations de QuadDist afin de créer un filtrage basé sur le même schéma que les consistances locales, mais avec une représentation des domaines par des poygones convexes. Autrement dit, utiliser les opérations de base sur les polygones (somme de Minkowski, intersections, unions, ...), pour calculer une approximation des contraintes de distance par des polygones . On obtiendrait ainsi un algorithme de filtrage global, très proche de celui de Heusch[51] mais plus facile à mettre en oeuvre. Se poserait encore une fois le problème de l'exactitude des approximations et donc de la garantie des calculs.

Techniques de résolution de CSPs continus Dans cette thèse, nous avons également exploré une technique de résolution spécifique aux contraintes de distance. Cette technique utilise des trous dans les domaines des variables que l'on identifie en utilisant la structure particulière des contraintes (cercle ou sphère). Dans un deuxième temps, nous avons généraliser cette approche à des contraintes quelconques.

- DS : **Décomposition Sémantique des contraintes de distance** Nous avons introduit une nouvelle méthode de recherche basée sur une décomposition des domaines des variables, qui exploite directement la sémantique des contraintes de distance. Les propriétés spécifiques de ces contraintes sont utilisées pour choisir des points de coupe plus intéressants que les choix par défaut des heuristiques standard. En fait, on peut identifier des trous dans les domaines en utilisant la structure des contraintes (cercles ou sphères). Le principe de DS est de modéliser le CSP continu par un CSP fini dont les valeurs sont des intervalles , puis d'utiliser une technique de résolution comme le maintien d'arc-consistance (MAC [112]) pour engendrer des sous-problèmes. Ces sous-problèmes sont 2B-consistants et vérifient des propriétés de monotonie et de convexité. Les résultats obtenus sur des problèmes académiques de taille réduite sont encourageants.
- MindTheGaps : **Une nouvelle stratégie de recherche pour les CSPs continus** Nous avons étendu la technique précédente à des CSPs non-linéaires quelconques. Cette stratégie stratégie de recherhche, nommée MindTheGaps, exploite les trous identifiés par les consistances locales et partielles. Ces trous indiquent des points de coupe dans les domaines qui permettent de réduire l'espace de recherche. Couper les domaines en utilisant ces trous permet d'élimininer certaines solutions redondantes et d'isoler plus facilement des solutions disjointes. Des résultats expérimentaux ont montré que pour beaucoup de problèmes, les performances de la recherche classique sont améliorées de manière significative en utilisant MindTheGaps.

  De futurs travaux pourront porter sur la mémorisation de trous. Il est possible que certains trous soient identifiés dans des branches différentes de l'arbre de recherche. L'idée est d'utiliser les trous que l'on a déjà identifié dans une branche précédente de la recherche pour décomposer le domaine courant lorsqu'aucun trou n'a été indentifié par le filtrage. D'autre part, la question du grand nombre de trous identifiés par les projections des fonctions trigonométriques reste en suspens.

De manière générale, l'idée d'exploiter les trous dans la résolution des contraintes numériques semble être une bonne voie à suivre. De futurs travaux pourraient porter sur d'autres techniques pour identifier des trous dans les domaines.

Annexe A

Nous démontrons dans cette annexe la propriété 5.2.1, que nous rappelons ici :

Proposition 1 *Soient trois points A, B et C et soit G le centre de gravité du triangle ABC. Alors, si u, v et w sont strictement positifs on a :*

$$\begin{array}{lcl} GA^2 = \frac{1}{9}(2u^2 + 2v^2 - w^2) & & AB = u \\ GB^2 = \frac{1}{9}(2u^2 + 2w^2 - v^2) & \Longleftrightarrow & AC = v \\ GC^2 = \frac{1}{9}(2v^2 + 2w^2 - u^2) & & BC = w \end{array}$$

Démonstration

$\underline{\Longleftarrow :}$

Soit ABC un triangle tel que $AB = u$, $AC = v$, et $BC = w$ et G son centre de gravité.
Soient I, J et K les milieux respectifs des segments $[AB]$, $[AC]$ et $[BC]$.
Soient $\alpha, \beta, \gamma \in [0, \pi]$, les angles non-orientés incidents respectivement aux sommets A, B et C (voir figure 1).
Deux propriétés élémentaire du centre de gravité vont nous servir dans cette preuve :

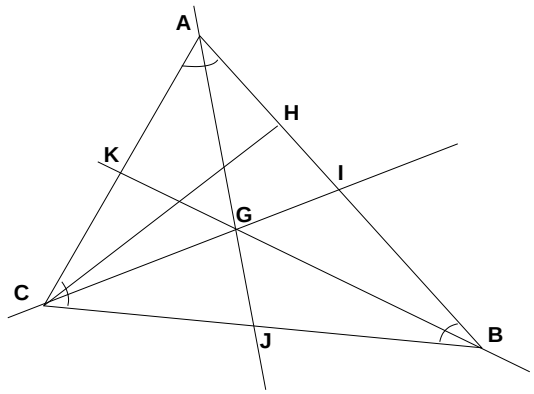


Fig. 1 – Un triangle ABC et son centre de gravité G

- G est l'intersection des médianes du triangle.
- $GC = \frac{2CI}{3}$, $GB = \frac{2BJ}{3}$ et $GA = \frac{2AK}{3}$.

On peut calculer le cosinus de l'angle α dans le triangle ABC en fonction des distances AB, AC et BC :

$$cos(\alpha) = \frac{AB^2 + AC^2 - BC^2}{2AB.AC}$$

De même, on peut calculer ce cosinus dans le triangles ABJ :

$$cos(\alpha) = \frac{AB^2 + AJ^2 - BJ^2}{2AB.AJ}$$

Si $AB \neq 0$ et $AC \neq 0$ alors avec ces deux égalités on obtient :

$$\begin{aligned}
\frac{AB^2+AC^2-BC^2}{2AB.AC} &= \frac{AB^2+AJ^2-BJ^2}{2AB.AJ} \\
\frac{AB^2+AC^2-BC^2}{2AB.AC} &= \frac{AB^2+\frac{AC^2}{4}-\frac{9GB^2}{4}}{AB.AC} \\
2AB^2 + 2AC^2 - 2BC^2 &= 4AB^2 + AC^2 - 9GB^2 \\
9GB^2 &= 2AB^2 + 2BC^2 - AC^2 \\
GB^2 &= \tfrac{1}{9}(2AB^2 + 2BC^2 - AC^2)
\end{aligned}$$

D'où $GB^2 = \frac{1}{9}(2u^2 + 2w^2 - v^2)$.
On peut obtenir les autres égalités en mettant en relation de la même façon les cosinus de β et γ dans les triangles ABC avec ACK et BCI. $\square$.

Ce qui est intéressant dans cette preuve est l'égalité des cosinus. Cela veut dire que la position dans l'espace du barycentre est liée aux angles du triangle. L'introduction de ce point va donc nous permettre de faire un meilleur filtrage, puisqu'il met en jeu implicitement la notion d'angle.

$\underline{\Longrightarrow :}$ Soit ABC un triangle et G son centre de gravité. Soient 3 réels positifs u, v et w tels que :

- $GA^2 = \frac{1}{9}(2u^2 + 2v^2 - w^2)$.
- $GB^2 = \frac{1}{9}(2u^2 + 2w^2 - v^2)$.
- $GC^2 = \frac{1}{9}(2v^2 + 2w^2 - u^2)$.

On a $AB^2 = \overrightarrow{AB}.\overrightarrow{AB} = (\overrightarrow{AG} + \overrightarrow{GB}).(\overrightarrow{AG} + \overrightarrow{GB})$. D'où,

$$2\overrightarrow{GA}.\overrightarrow{GB} = AG^2 + GB^2 - AB^2 \tag{1}$$

De même, on obtient :

$$2\overrightarrow{GA}.\overrightarrow{GC} = AG^2 + GC^2 - AC^2 \tag{2}$$

$$2\overrightarrow{GB}.\overrightarrow{GC} = GB^2 + GC^2 - BC^2 \tag{3}$$

G étant l' isobarycentre de A, B et C, on a l'égalité vectorielle : $\overrightarrow{GA} + \overrightarrow{GB} + \overrightarrow{GC} = \overrightarrow{0}$.
Or, en faisant le produit scalaire successivement par $\overrightarrow{GA}$, $\overrightarrow{GB}$, et $\overrightarrow{GC}$ des 2 côtés de cette égalité on obtient :

$$2GA^2 + 2\overrightarrow{GA}.\overrightarrow{GB} + 2\overrightarrow{GA}.\overrightarrow{GB} = 0$$

$$2GB^2 + 2\overrightarrow{GA}.\overrightarrow{GB} + 2\overrightarrow{GB}.\overrightarrow{GC} = 0$$

$$2GC^2 + 2\overrightarrow{GA}.\overrightarrow{GC} + 2\overrightarrow{GB}.\overrightarrow{GC} = 0$$

En introduisant les équations 1, 2 et 3 et après simplifications, on obtient le système suivant :

$$4GA^2 + GB^2 + GC^2 = AB^2 + AC^2$$

$$4GB^2 + GA^2 + GC^2 = AB^2 + BC^2$$

$$4GC^2 + GA^2 + GB^2 = AC^2 + BC^2$$

Enfin, en résolvant ce système linéaire dont les inconnues sont AB^2, AC^2, et BC^2, puis en remplaçant les distances GA, GB et GC par leur expression en fonction de u, v, et w on obtient :

$$AB^2 = 2GA^2 + 2GB^2 - GC^2 = u^2$$

$$AC^2 = 2GA^2 + 2GC^2 - GB^2 = v^2$$

$$BC^2 = 2GB^2 + 2GC^2 - GA^2 = w^2$$

D'où, finalement :

$$AB = u \qquad AC = v \qquad BC = w$$

□.

Remarque *L'hypothèse que l'on fait sur les distances AB, AC et BC, garantit que le domaine des distances GA, GB et GC est calculable. En effet, on a :*
$0 \leq |AB - AC| \leq BC \leq AB + AC$
$\Rightarrow |AB - AC|^2 \leq BC^2 \leq (AB + AC)^2$
$\Leftrightarrow AB^2 + AC^2 - 2.AB.AC \leq BC^2 \leq AB^2 + AC^2 + 2.AB.AC$
$\Leftrightarrow -(AB^2 + AC^2) - 2.AB.AC \leq -(2AB^2 + 2AC^2) + BC^2 \leq -(AB^2 + AC^2) + 2.AB.AC$
$\Leftrightarrow \frac{1}{9}(AB - AC)^2 \leq GA^2 \leq \frac{1}{9}(AB + AC)^2$
Autrement dit, le domaine de GA^2 est encadré par deux intervalles positifs ou nuls, et donc le domaine de GA est calculable en utilisant l'arithmétique des intervalles. On peut évidemment trouver le même type d'encadrement pour GB^2 et GC^2.

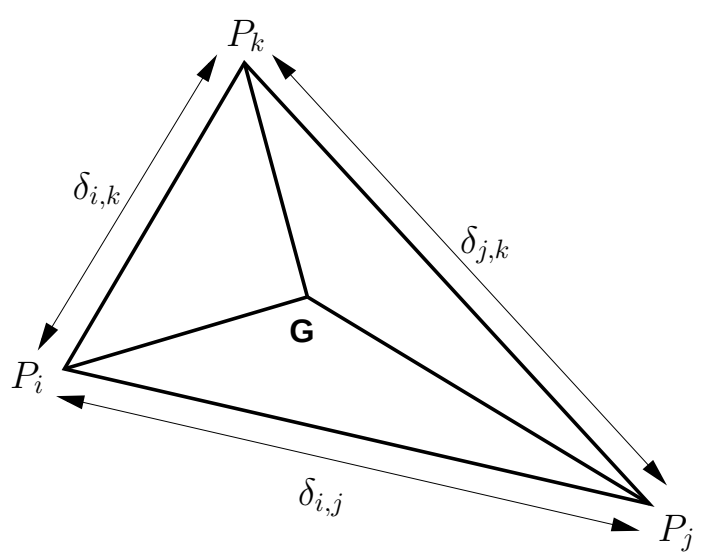


FIG. 2 – Introduction du centre de gravité dans le triangle $\mathcal{P}_i\mathcal{P}_j\mathcal{P}_k$

Bibliographie

[1] G. Alefeld and J. Hertzberger. *Introduction to Interval Computation.* Academic Press, New York, 1983.

[2] H. Batnini. Contraintes globales pour la résolution de contraintes de distance. Master's thesis, Université de Nice Sophia-Antipolis, Juin 2002.

[3] H. Batnini. Introduction of redundant constraints for solving distance constraints systems. *Journal of the university of Saärbrück, CALCULEMUS autumn school : Student poster abstracts, Pisa, Italy*, pages 19–23, September 2002.

[4] H. Batnini, C. Michel, and M. Rueher. Mind the gaps : A new splitting strategy for consistency techniques. In *CP*, pages 77–91, Sitges, Barcelona, Spain, October 2005.

[5] H. Batnini and M. Rueher. Filtrage local par décomposition de csp continus. In *Actes JNPC'03. 9eme Journées Nationales pour la résolution de Problèmes NP-complets*, pages 39–51, Juin 2003.

[6] H. Batnini and M. Rueher. Semantic decomposition for solving distance constraint. In F. Rossi, editor, *Proc. of CP'03 : 9th International Conference on "Principles and Practice of Constraint Programming")*, LNCS 2833, pages 964–964. Springer Verlag, September 2003.

[7] H. Batnini and M. Rueher. Décomposition sémantique pour la résolution de systèmes d'équations de distances. *JEDAI(Journal Electronique d'Intelligence Artificielle)*, 2(1), 2004. Édition spéciale JNPC 2003.

[8] H. Batnini and M. Rueher. Quaddist : Filtrage global pour les contraintes de distance. In *Actes JNPC'04(10èmes Journées Nationales pour la résolution pratique de Problèmes NP-complets)*, pages 59–71, Angers, Juin 2004.

[9] H. Batnini, M. Rueher, and C. Michel. Une stratégie de résolution orientée par la topologie des csp numériques. In *Actes JFPC'05 (1ères Journées Francophones de Programmation par Contraintes)*, pages 199–210, Lens, 2005.

[10] N. Beldiceanu, M. Carlsson, and J-X. Rampon. Tr2005-06 : global constraint catalog. Technical report, Swedish Institue of Computer Science, 2005.

[11] N. Beldiceanu, M. Carlsson, J-X. Rampon, and C. Truchet. Graph invariants as necessary conditions for global constraints. In P. Van Beek, editor, *Proc. of CP'05 : 11th International Conference on "Principles and Practice of Constraint Programming")*, LNCS 3709, pages 92–106, September, publisher = Springer Verlag 2005.

[12] N. Beldiceanu and E. Contjean. Introducing global constraints in chip. *Journal of Mathematical and Computer Modelling*, 12 :97–123, 1994.

[13] F. Benhamou, F. Goualard, L. Granvilliers, and J.F. Puget. Revising hull and box consistency. In *International Conference on Logic Programming*, pages 230–244, 1999.

[14] F. Benhamou, D. McAllister, and P. Van Hentenryck. CLP(intervals) revisited. In Maurice Bruynooghe, editor, *Proceedings of the 1994 International Symposium*, pages 124–138. MIT Press, 1994.

[15] F. Benhamou and W. Older. Applying interval arithmetic to real, integer and Boolean constraints. *Journal of Logic Programming*, 32(1) :1–24, 1997.

[16] C. Bessière. Arc-consistency and arc-consistency again. *Artif. Intell.*, 65(1) :179–190, 1994.

[17] C. Bessière and J.C Régin. Enforcing arc consistency on global constraints by solving subproblems on the fly. In *PPCP*, pages 103–117, 1999.

[18] J.R. Bitner and E.M. Reingold. Backtrack programming techniques. *Commun. ACM*, 18(11) :651–656, 1975.

[19] C. Bliek, B. Neveu, and G. Trombettoni. Using graph decomposition for solving continuous csps. In M. Maher and J-F. Puget, editors, *Proc. of CP'98 : 4th International Conference on "Principles and Practice of Constraint Programming"*, LNCS 1520, pages 102–116, Pisa, Italy, November 1998. Springer Verlag.

[20] L.M. Blumenthal. *Theory and Application of Distance Geometry.* Oxford University Press, 1953.

[21] L. Bordeaux, E. Monfroy, and F. Benhamou. Raisonnement sur les propriétés de contraintes numériques. In M. Rueher, editor, *Programmation en logique par contrainte*, pages 13–26. JFPLC'02, Hermès Science publications, 2002.

[22] H. Brönninmann and S. Pion. Exact rounding for geometric constructions. In *Proc. of International symposium on Scientific Computing, Computer Arithmetic and Validated Numerics(SCAN)*, 1997.

[23] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.* PhD thesis, Université d'Innsbruck, Autriche, 1965.

[24] B. Buchberger. Gröbner bases : an algorithmic method in polynomial ideal theory. In N.K. Bose, editor, *Recent trends in multidimensional system theory.* Reidel publ. comp., 1985.

[25] M. Ceberio. *Contribution à l'étude des CSPs numériques sous et sur-contraints. Outils symboliques et contraintes flexibles continues.* PhD thesis, Université de Nantes, 2003.

[26] H. Collavizza, F. Delobel, and M. Rueher. A note on partial consistencies over continuous domains. In M. Maher and J-F. Puget, editors, *Proc. of CP'98 : 4th International Conference on "Principles and Practice of Constraint Programming"*, LNCS 1520, pages 147–162, Pisa, Italy, November 1998. Springer Verlag.

[27] H. Collavizza, F. Delobel, and M. Rueher. Comparing partial consistencies. *Journal of Reliable Computing*, 5 :213–228, 1999.

[28] H. Collavizza, F. Delobel, and M. Rueher. Extending consistent domains of numeric csp. In *Proc of IJCAI'99 : 16th International joint conference on artificial intelligence*, volume 1, pages 406–411, Stockholm, Sweden, August 1999.

[29] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41(1) :89–95, 1989.

[30] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms*. The MIT Press, 1991.

[31] G.M. Crippen and T.F Havel. *Distance geometry and molecular conformation*. John Wiley and sons, 1988.

[32] R. Dechter. Learning while searching in constraint-satisfaction-problems. In *AAAI*, pages 178–185, 1986.

[33] R. Dechter. Enhancement schemes for constraint processing : Backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41(3) :273–312, 1990.

[34] R. Dechter. From local to global consistency. *Artif. Intell.*, 55(1) :87–108, 1992.

[35] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *IJCAI*, pages 271–277, 1989.

[36] A.K. Dhingra, A.N. Almadi, and D. Kohli. A gröbner-sylvester hybrid method for closed-form displacement analysis of mechanisms. *Journal of Mechanical Design*, 122 :431–438, December 2000.

[37] P. Dietmaier. The stewart-gough platform of general geometry can have 40 real postures. In J. Lenarcic and M.L. Husty, editors, *Advances in Robot Kinematics : Analysis and Control*, pages 7–16. Kluwer Academic Publisher, 1998.

[38] L. Doherty, K. S.J. Pister, and L. El Ghaoui. Convex position estimation in wireless sensor networks. In *Proc. of the IEEE Infocom*, pages 1655–1663, Alaska, April 2001. IEEE.

[39] I.Z. Emiris and B. Mourrain. Polynomial system solving : the case of a six-atom molecule. Technical Report RR-3075, 1996.

[40] J.-C. Faugère. A new efficient algorithm for computing gröbner bases (f_4). *Journal of Pure and Applied Algebra*, 139(1-3) :61–88, 1999.

[41] E.C. Freuder. Synthesizing constraint expressions. *Communication of the ACM*, 21(11) :958–966, Nov. 1978.

[42] E.C. Freuder. Backtrack-free and backtrack-bounded search. pages 343–369, 1988.

[43] J. Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In *IJCAI*, page 457, 1977.

[44] S.W. Golomb and L.D. Baumert. Backtrack programming. *Journal of ACM*, 12(4) :516–524, 1965.

[45] C. Gosselin, J. Sefrioui, and M. J. Richard. Polynomial solutions to the direct kinematic problem of planar three degree of freedom parallel manipulators. *Mechanism and Machine Theory*, 27(2) :107–119, 1992.

[46] L. Granvilliers. Realpaver : Solving non linear constraints by interval computations. User's manual, July 2003. http ://www.sciences.univ-nantes.fr/info/perso/permanents/granvil/realpaver.

[47] D.U. Greemwald. *Programmation linéaire et algorithme du simplexe*. Dunod, 1960.

[48] E. Hansen and R. Greenberg. An interval newton method. *Applied Mathematics and Computations*, 12 :89–98, 1983.

[49] E.R. Hansen. *Global optimization using interval analysis.* Marcel Deckler, 1992.

[50] R.M. Haralick and G.L. Elliott. Increasing tree search eciency for constraint satisfaction problems. *Artificial Intelligence*, 14 :263–313, 1980.

[51] M. Heusch. distn : An euclidean distance global constraint. In F. Rossi, editor, *Proc. of CP'03 : 9th International Conference on "Principles and Practice of Constraint Programming")*, LNCS 2833, pages 975–975. Springer Verlag, September 2003.

[52] M.L. Husty. An algorithm for solving the direct kinematic of stewart-gough-type platforms. *Mechanism and Machine Theory*, 4(31) :365–380, 1996.

[53] E. Hyvönen. Constraint reasoning based on interval arithmetic : the tolerance propagation approach. *Artificial Intelligence*, 58(1) :71–112, December 1992.

[54] ILOG. *Solver Reference manual http ://www.ilog.com/product/jsolver.* 2002.

[55] C. Innocenti. Forward kinematics in polynomial form of the general stewart platform. *ASME Journal of Mechanical Design*, 2(123) :254–260, June 2001.

[56] Faugère J-C and Lazard D. The combinatorial classes of parallel manipulators. *Mechanism and Machine Theory*, 6(30) :765–776, 1995.

[57] Merlet J-P. Solving the forward kinematics of a gough-type parallel manipulator with interval analysis. *International Journal of Robotics Research*, 3(23) :221–236, 2004.

[58] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis.* Springer, 2001.

[59] C. Jermann. *Résolution de contraintes géométriques par rigidification récursive et propagation d'intervalles.* PhD thesis, Université de Nice Sophia Antipolis, 2002.

[60] C. Jermann, B. Neveu, and G. Trombettoni. A new structural rigidity for geometric constraints systems. In *Proc. of Fourth International Workshop on Automated Deduction in Geometry(ADG'02)*, Linz, Austria, 2002.

[61] C. Jermann, B. Neveu, and G. Trombettoni. Inter-block backtracking : Exploiting the structure in continuous csps. In *2nd International Workshop on Global Constrained Optimization and Constraint Satisfaction COCOS'03*, Lausanne, Switzerland, 2003.

[62] C. Jermann, G. Trombettoni, B. Neveu, and M. Rueher. A constraint programming approach for solving rigid geometric systems. In *Proc. of CP'00 : Sixth International Conference on "Principles and Practice of Constraint Programming"*, LNCS 1894, pages 233–248, Singapore, September 2000. Springer Verlag.

[63] P. Jégou. *Contribution à l'étude des problèmes de satisfaction de contraintes : algorithmes de propagation et de résolution ; propagation de contraintes dans les réseaux dynamiques.* PhD thesis, CRIM, USTL, Université de Montpellier, 1991.

[64] Narendra Jussien and Olivier Lhomme. Dynamic domain splitting for numeric CSPs. In *European Conference on Artificial Intelligence*, pages 224–228, 1998.

[65] W.M. Kahan. A more complete interval arithmetic. Technical report, University of Toronto, Canada, 1968.

[66] R.B. Kearfott. A review of techniques in the verified solution of constrained global optimization problems. In R. Baker Kearfott and Vladik Kreinovich, editors, *Applications of Interval Computations*, pages 23–59. Kluwer, Dordrecht, Netherlands, 1996.

[67] R.B. Kearfott. *Rigorous global search : continuous problems.* Kluwer, 1996.

[68] I. Kotsireas and I. Lazard. Central configurations of the 5-body problem with equal masses in three dimensional space. In *Proceedings of CASC*, 1998.

[69] L. Krippahl and P. Barahona. Psico : Combining constraint programming and optimisation to solve macromolecular structures. In *Proc. of ERCIM/COMPULOG Workshop on Constraints*, Univ. of Padova, Italy, June 2000.

[70] L. Krippahl and P. Barahona. Propagating n-ary rigid-body constraints. In F. Rossi, editor, *Proc. of CP'03 : 9th International Conference on "Principles and Practice of Constraint Programming")*, LNCS 2833, pages 452–465. Springer Verlag, September 2003.

[71] V. Kumar. Algorithms for constraint-satisfaction problems : A survey. *AI Magazine*, 13(1) :32–44, 1992.

[72] D. Lazard. Stewart platforms and gröbner basis. In *Proceedings of Advances in Robotics Kinematics*, pages 136–142, Septembre 1992.

[73] D. Lazard. Generalized stewart platform : How to compute with rigid motions ? In *IMACS Symp. on Symbolic Computation*, pages 85–88, 1993.

[74] Y. Lebbah. *Contribution à la résolution de contraintes par consistance forte*. Thès de doctorat, École des Mines de Nantes, 1999.

[75] Y. Lebbah, Michel C., M. Rueher, D. Daney, and J.P. Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*. Accepted for publication.

[76] Y. Lebbah, M. Rueher, and C. Michel. A global filtering algorithm for handling systems of quadratic equations and inequations. In P. Van Hentenryck, editor, *Proc of CP'02 : 8th International Conference on "Principles and Practice of Constraint Programming"*, LNCS 2470, Cornell University, Ithaca, NY, USA, September 2002. Springer Verlag.

[77] T-Y Lee and J-K. Shim. Forward kinematics of the general 6-6 stewart platform using algebraic elimination. *Mechanism and Machine Theory*, 36(9) :1073–1085, September 2001.

[78] O. Lhomme, , A. Gotlieb, and M. Rueher. Dynamic optimization of interval narrowing algorithms. *Journal of Logic Programming(Elsevier Science Inc).*, 37(1-3) :165–183, 1998.

[79] O. Lhomme. Consistency techniques for numerical csps. In *IJCAI-93*, pages 232–238, 1993.

[80] O. Lhomme. *Contribution à la résolution de contraintes sur les réels par propagation d'intervalles*. Thèse de doctorat, Université de Nice-Sophia Antipolis, 1994.

[81] A-X. Liu and T-L. Yang. Configuration analysis of a class of parallel structures using improved continuation. In *9th World Congress on the Theory of Machines and Mechanisms*, pages 155–158, Milan, September 1995.

[82] F.E.A. Lucas. *Récréations Mathématiques*, volume 1-4. Gautier-Villars, Paris, 1891. Réédité depuis par Blanchard, Paris, 1959.

[83] A.K. Mackworth. Constraint satisfaction. In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, New York, 1987. Wiley.

[84] A.K Macworth. Consistency in networks of relations. *Artificial Intelligence*, pages 99–118, 1977.

[85] M. Cs. Markót. An interval method to validate optimal solutions of the "packing circles in a unit square". *Problems, Central European Journal of Operational Research*, 8 :63–78, 2000.

[86] M. Cs. Markót and T. Csendes. A new verified optimization technique for the "packing circles in a unit square" problems. *SIAM*, 2004. (submitted).

[87] J-P. Merlet. Solving the forward kinematics of gough-type parallel manipulator with interval analysis. *Research report INRIA 2003*, RR-4013, 2003.

[88] P. Meseguer. Constraint satisfaction problems : An overview. *AI Commun.*, 2(1) :3–17, 1989.

[89] C. Michel, Y. Lebbah, and M. 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 2003*, pages 210–220, Université de Montréal, 2003. CRT.

[90] R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artif. Intell.*, 28(2) :225–233, 1986.

[91] U. Montanari. Networks of constraints : Fundamental properties and applications to image processing. *Information science*, 7 :95–132, 1974.

[92] R. Moore. *Interval analysis*. Prentice-Hall, 1966.

[93] B. Mourrain. The 40 g̈enericp̈ositions of a parallel robot. In *Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation(ISSAC)*, pages 173–182. ACM Press, 1993.

[94] B. A. Nadel. Tree search and arc consistency in constraint satisfaction algorithms. pages 287–342, 1988.

[95] A. Neumaier. *Interval Methods for Systems of Equations*, volume 37. Encyclopedia of Mathematics and its Applications, 1990.

[96] W. Older and F. Benhamou. Programming in CLP(BNR). In *Position Papers for the First Workshop on Principles and Practice of Constraint Programming*, pages 239–249, Newport, RI, USA, 1993.

[97] J.M. Porta, L. Ros, F. Thomas, and C Torras. A branch-and-prune algorithm for solving systems of distance constraints. In *IEEE Conference on Robotics and Automation*, Taipei, Taiwan, Sept 2003.

[98] Jean-François Puget. A fast algorithm for the bound consistency of alldiff constraints. In *AAAI '98/IAAI '98 : Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 359–366, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[99] J.F. Puget and P. Van Hentenryck. A constraint satisfaction approach to a circuit design problem. Technical Report CS-96-34, 1996.

[100] J.F. Puget and P.. Van Hentenryck. A constraint satisfaction approach to a circuit design problem. *Journal of Global Optimization*, 13 :75–93, 1998.

[101] P.W. Jr. Purdom. Search rearrangement backtracking and polynomial average time. *Artif. Intell.*, 21(1-2) :117–133, 1983.

[102] M. Raghavan. The stewart platform of general geometry has 40 configurations. In *ASME Design and Automation Conf.*, volume 32, pages 397–402, 1992.

[103] D. Ratz. Box-splitting strategies for the interval Gauss–Seidel step in a global optimization method. *Computing*, 53 :337–354, 1994.

[104] J-C. Régin. A filtering algorithm for constraints of difference in csps. In *AAAI '94 : Proceedings of the twelfth national conference on Artificial intelligence (vol. 1)*, pages 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[105] J-C. Régin. Generalized arc consistency for global cardinality. In *AAAI-96*, pages 209–215, 1996. Oregon.

[106] J-C. Régin. The symetric alldiff constraint. In *IJCAI-99*, pages 420–425, 1999.

[107] J-C. Régin. Ac-* : A configurable, generic and adaptative arc consistency algorithm. In P. Van Beek, editor, *Proc. of CP'05 : 11th International Conference on "Principles and Practice of Constraint Programming")*, LNCS 3709, pages 505–519, September, publisher = Springer Verlag 2005.

[108] J-C. Régin and M. Rueher. A global constraint combining sum and difference constraints. In *Proc. of CP'00 : Sixth International Conference on "Principles and Practice of Constraint Programming"*, LNCS 1894, pages 384–396, Singapore, September 2000. Springer Verlag.

[109] F. Ronga and T. Vust. Stewart platforms without computer ? In *Conf. Real Analytic and Algebraic Geometry*, pages 197–212, Trento, 1992.

[110] F. Rossi, Petrie C.J., and V. Dhar. On the equivalence of constraint satisfaction problems. In *ECAI*, pages 550–556, 1990.

[111] F. Rouiller. Real roots counting for some robotics problems. *Computational Kinematics*, 1995.

[112] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A. Borning, editor, *Proc. of PPCP'94 : Second International Workshop on "Principles and Practice of Constraint Programming"*, LNCS 874, pages 10–20. Springer Verlag, 1994.

[113] D. Sam-Haroud. *Constraint consistency techniques for continuous domains.* PhD thesis, École polytechnique fédérale de Lausanne, 1995.

[114] S.V. Sreenivasan and P. Nanua. Solution of the direct position kinematics problem of the general stewart platform using advanced polynomial continuation. In *22nd Biennial Mechanisms Conf.*, pages 99–106, Scottsdale, September. 1992.

[115] P.G. Szabó, T. Csendes, L.G. Casado, and I. García. Equal circles packing in a square i. - problem setting and bounds for optimal solutions. *Optimization Theory - Recent Developments from Matrahaza. Kluwer, Dordrecht*, pages 191–206, 2001.

[116] C. Traverso. The posso test suite examples, 1993. Available at http ://www.inria.fr/saga/POL/index.html.

[117] M.R. Van Dongen. Ac-3d an efficient arc-consistency algorithm with a low space complexity. In P. Van Hentenryck, editor, *Proc of CP'02 : 8th International Conference on "Principles and Practice of Constraint Programming"*, LNCS 2470, Cornell University, Ithaca, NY, USA, September 2002. Springer Verlag.

[118] P. Van Hentenryck and Y. Deville. The cardinality operator : A new logical connective for constraint logic programming. In *ICLP*, pages 745–759, 1991.

[119] P. Van Hentenryck, D. McAllister, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM, Journal of Numerical Analysis*, 34(2) :797–827, April 1997.

[120] D. Waltz. Understanding line drawings of scenes with shadows. In Patrick Henry Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975. (D'abord publié dans un technical report du MIT en 1972).

[121] Y. Zhang and R.H.C. Yap. Making ac-3 an optimal algorithm. In *IJCAI*, pages 316–321, 2001.