UNIVERSITE D'ANGERS **Année** 2005

N° d'ordre 736

Propagation de contraintes sur les intervalles Application à l'étalonnage des robots

Thèse de doctorat

Spécialité : Automatique Et Informatique Appliquee.

ECOLE DOCTORALE D'ANGERS

Présentée et soutenue publiquement

le Jeudi 8 Décembre 2005 **à** ANGERS **par**

Xavier BAGUENARD

Devant le jury ci-dessous

Wisama KHALIL,	Président du jury	Professeur, IRCCyN, École Centrale de NANTES.
Laurent GRANVILLIERS,	Rapporteur	Professeur, LINA, Université de NANTES.
Jean Pierre MERLET,	Rapporteur	Directeur de recherche, INRIA SOPHIA ANTIPOLIS.
Laurent HARDOUIN,	Examinateur	Professeur, LISA, ISTIA ANGERS.
Luc JAULIN,	Examinateur	Professeur, E3I2, ENSIETA BREST.
Philippe LUCIDARME,	Examinateur	Maître de conférences, LISA, IUT ANGERS.

LABORATOIRE D'INGENIERIE DES SYSTEMES AUTOMATISES

62, Avenue Notre Dame du Lac, 49000 Angers ED363

Résumé

Dans cette thèse, une méthode ensembliste utilisant la propagation de contraintes sur les intervalles est proposée pour l'étalonnage géométrique d'un robot à 6 degrés de libertés. Cette méthode permet d'enfermer chaque paramètre à estimer à l'intérieur d'un intervalle dont la largeur dépend de l'incertitude sur les mesures collectées, des erreurs de modélisation et du pessimisme inhérent aux méthodes ensemblistes.

Le problème de l'étalonnage géométrique est présenté et formalisé comme un problème de satisfaction de contraintes. Mises sous la forme de graphe acyclique orienté, les contraintes sont ensuite projetées sur les domaines afin de determiner un intervalle le plus petit possible pour chacun des paramètres recherchés.

La plus petite boîte encadrant l'ensemble des solutions n'étant pas forcément obtenue, un ajout de contraintes redondantes liées au problème ou aux dérivées des contraintes est proposé afin d'améliorer les résultats obtenus par la propagation.

Mots clés : analyse par intervalles, calcul ensembliste, estimation ensembliste, étalonnage des robots séries, graphe acyclique orienté (DAG), méthodes garanties, problème de satifaction de contraintes (CSP), propagation de contraintes, robotique.

Abstract

Geometric calibration is used to improve positioning accuracy of industrial robots. This thesis presents a method using interval constraint propagation applied to the kinematic calibration of a six degrees of freedom robot. For each parameter, an interval is computed by the method to enclose the actual value for the parameter in spite of uncertainties on measures, model errors and pessimism.

Geometric calibration problem can be cast into a constraints satisfaction problem involving variables, domains and constraints. These constraints are then projected on the domains for the parameters in order to determine the smallest interval for each variable.

Unfortunately, the smallest box enclosing the solution set is not always obtained. The addition of redundant constraints generated automatically by derivation made it possible to improve the contractions of the domains.

Key-Words : interval analysis, set computation, set estimation, serial robots calibration, directed acyclic graph (DAG), reliable methods, constraints satisfaction problem (CSP), constraints propagation, robotics.

Remerciements

Mes premiers remerciements sont pour Luc JAULIN. Aujourd'hui en cette fin de thèse je me rends compte du chemin parcouru et du chemin qu'il me reste à parcourir. Merci à toi pour ce bout de chemin que nous avons fait ensemble.

Merci à Philippe LUCIDARME, qui en cette fin de thèse, a eu un rôle déterminant. J'avoue que sans toi cette thèse ne se serait jamais finie à cette date.

Merci à l'ensemble des membres du jury, merci à Wisama KHALIL pour son accueil lors des contacts et des discussions concernant la robotique, merci à Jean Pierre MERLET et Laurent GRANVILLIERS pour la relecture détaillée de ma thèse, enfin merci à Laurent HARDOUIN pour la relecture de la thèse et la discussion que nous avons eu sur cette thèse.

Un grand merci à tous les membres du LISA et de l'ISTIA avec une spéciale dédicace à tous ceux qui ont eu à me supporter dans la salle E37, aux anciens de la salle Mehdi, Didi, Mounir, Bahdi... à ceux que j'embête encore Sébastien, Laurent, Iteb, Nicolas, Massa, Michel et Samir. Un grand merci à vous tous pour tout ce que vous m'avez apporté.

Merci aux personnes qui me sont proches, l'ensemble de la famille, mes parents, et surtout à ma petite famille, Caroline qui a supporté des intervalles pendant ces 4 années et Florian qui m'a permis d'avoir le mois le plus rempli de ma vie.

Une nouvelle fois je le redis pour toi Caroline, merci !

Table des matières

Chapitre 1

Introduction

1.1 Méthodes par intervalles

Les méthodes par intervalles [57] ont connu de nombreux développements pendant les 40 dernières années avec des succès importants dans la démonstration assistée par ordinateur de conjecture et l'optimisation globale (voir [1, 40] pour une présentation des applications).

Un premier avantage de ces méthodes est le fait de palier au codage fini des réels des ordinateurs permettant de garantir à ces méthodes la complétude. La méthode est alors dite complète car elle ne perd aucune solution. L'inconvénient associé est l'obtention d'un intervalle dont la taille peut varier laissant l'utilisateur devant un choix qui parfois le déroute.

Le deuxième avantage correspond à la possibilité de démontrer que l'ensemble des valeurs prises dans un intervalle sont solutions ou non solutions. Cette possibilité de démonstration quelque soit le système d'équation du problème (non linéaire,...) étend le champ des démonstrations possibles par ordinateur (détermination de propriétés de connexité sur des ensembles, génération de preuves,...). Dans la communauté automatique, cette fonctionnalité permet alors d'aborder les problèmes difficiles que sont la détection de défauts, la commande robuste,...

Mais les algorithmes de découpage (branch and bound) liés aux méthodes par intervalles se heurtent au problème de la complexité exponentielle, qui pour des problèmes de dimension >5, devient relativement important. Grâce à l'apport de la communauté de programmation par contraintes sur les entiers, ces limites ont été repoussées en réalisant un filtrage sur les domaines intervalles afin de supprimer des parties non solution (branch and prune). Les algorithmes de propagation de contraintes permettent d'obtenir une boîte encadrant l'ensemble de solutions. Ils permettent aussi de réduire non pas la complexité de l'algorithme mais son temps de calcul dans la majorité des cas.

Les problèmes de grande dimension restent difficilement abordables via ces méthodes. Constatant cet état de fait, nous nous sommes concentrés non plus sur la caractérisation de la totalité de l'ensemble de solutions mais sur la recherche d'informations utilisables pour nos applications.

La recherche de la plus petite boîte encadrant l'ensemble de solutions est alors devenue notre objectif. Les applications et en particulier l'étalonnage géométrique des robots séries nous guidant pour ajuster nos méthodes aux solutions recherchées.

1.2 Etalonnage géométrique des robots séries

Positionner les manipulateurs avec une très grande précision est essentiel dans l'ensemble des tâches qu'effectuent les robots (que ce soit dans l'industrie, en électronique, en médecine,...). L'étalonnage de robots permet de renforcer cette précision en ajustant le modèle géométrique aux déplacements réels du robot. Dans les faits, les méthodes d'étalonnage recherchent les valeurs des paramètres géométriques (longueur de bras, angle de construction, offset, paramètres d'installation du robot, de l'outil...). Ces paramètres, quand cela est possible, sont ensuite integrés au contrôleur permettant de générer le mouvement du robot et ainsi de réduire les erreurs de positionnement.

L'architecture des robots séries a aussi une influence sur l'erreur de positionnement qui peut être importante, ce qui rend l'étalonnage indispensable. En effet une erreur sur la première articulation engendrera un décalage sur l'ensemble des articulations suivantes. Les robots parallèles quand à eux ont plusieurs bras ou chaînes cinématiques qui soutiennent la plateforme et l'outil. L'erreur sur un paramètre de ces chaînes sera donc moyennée par le fait que plusieurs bras maintiennent la plateforme (voir figure 1-1).

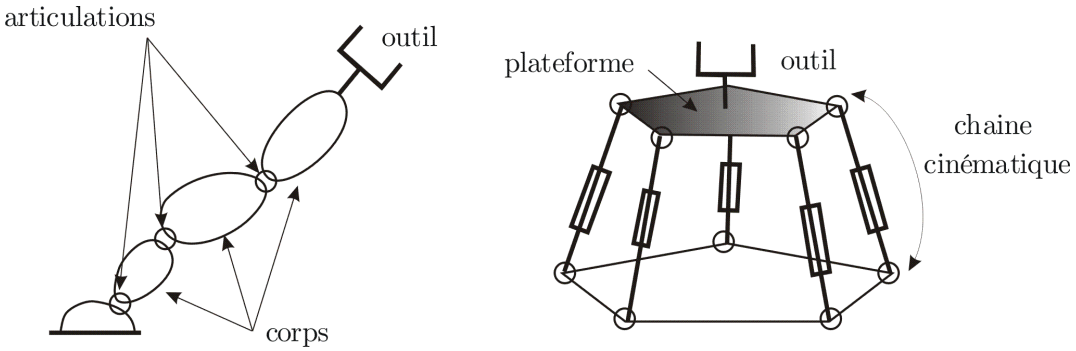

FIG. 1-1 – Présentation des robots séries et parrallèles.

La modélisation joue aussi un rôle dans la performance des méthodes d'étalonnage. Le modèle géométrique direct est facile à obtenir pour les robots séries. Pour une configuration des articulations motorisées, le modèle donne un et un seul point de l'espace de travail du robot. En utilisant les équations du modèle géométrique direct, les méthodes classiques réalisent une minimisation pour effectuer l'étalonnage.

Puisque les méthodes classiques donnent d'assez bons résultats sur l'étalonnage des robots séries, pourquoi utiliser les méthodes par intervalles ? L'objectif, dans un premier temps, a été pour

nous de comparer les méthodes par intervalles aux méthodes existantes. En effet les méthodes classiques donnent un point solution mais très peu d'informations sur l'incertitude autour de ce point. L'intérêt est alors d'étudier ce que les méthodes par intervalles peuvent apporter.

Le problème de l'étalonnage d'un robot série a donc été formalisé pour appliquer les méthodes par intervalles.

1.3 Le formalisme CSP

Pour l'ensemble des problèmes que nous allons traiter, l'utilisation du formalisme de la communauté de programmation par contraintes permet de modéliser le problème pour se concentrer sur la méthode de résolution. Chacun des problèmes peut alors être généré pour entrer dans le formalisme CSP.

Un *Problème de Satisfaction de Contraintes* ou CSP est défini par trois ensembles : l'ensemble des variables $\mathcal{V}$ relatif au problème, l'ensemble des domaines continus $\mathcal{D}$ associés à ces variables et l'ensemble des contraintes $\mathcal{C}$ reliant les variables entre elles. Nous pouvons donc l'écrire sous la forme :

$$\mathcal{CSP} : \left\{ \begin{array}{lll} \mathcal{V} : & \{x_1, x_2, \ldots & , x_n \\ & \quad y_1, y_2, \ldots & , y_m\}, \\ \mathcal{D} : & \{[x_1], [x_2], \ldots & , [x_n] \\ & \quad [y_1], [y_2], \ldots & , [y_m]\}, \\ \mathcal{C} : & f_1(\mathbf{x}) = y_1 & \\ & \ldots & \\ & f_m(\mathbf{x}) = y_m & \end{array} \right\} \tag{1.1}$$

ou encore en prenant la notation vectorielle :

$$\mathcal{CSP} : \left\{ \begin{array}{ll} \mathcal{V} : & \{\mathbf{x}, \mathbf{y}\}, \\ \mathcal{D} : & \{[\mathbf{x}], [\mathbf{y}]\}, \\ \mathcal{C} : & \{\mathbf{f}(\mathbf{x}) = \mathbf{y}\}. \end{array} \right\}. \tag{1.2}$$

Associé à ce CSP nous définissons l'ensemble de solutions :

$$\mathbb{S} \triangleq \{\mathbf{x} \in [\mathbf{x}] \mid \mathbf{f}(\mathbf{x}) \in [\mathbf{y}]\}. \tag{1.3}$$

Nous avons choisi, dans le formalisme, de séparer deux types de variables : les variables $\mathbf{x}$ et les variables $\mathbf{y}$. Ce choix est lié à la classe de problèmes que nous rencontrons en automatique et robotique qui se formalise par les contraintes non linéaires :

$$\mathbf{f}(\mathbf{x}) = \mathbf{y}. \tag{1.4}$$

Ce choix sera discuté dans le chapitre 4 sur les méthodes de résolution.

Pour les problèmes de grande dimension, caractériser l'ensemble $\mathbb{S}$ devient difficile, nous avons donc choisi de déterminer une boîte, si possible la plus petite, l'encadrant avec des méthodes de propagation de contraintes sur les intervalles.

Prenons un exemple simple pour illustrer notre approche. Le CSP $\mathcal{H}_1$ défini par :

$$\mathcal{H}_1 : \left\{ \begin{array}{lcl} \mathcal{V}_1 & = & \{x_1, x_2, y_1, y_2\}, \\ \mathcal{D}_1 & = & \{[x_1] =]-\infty; +\infty[, [x_2] =]-\infty; +\infty[, \\ & & \quad [y_1] = [1,2], [y_2] = [0,1], \}, \\ \mathcal{C}_1 & = & \{y_1 = x_1^2 + x_2^2, \quad y_2 = x_1 - x_2\}. \end{array} \right\}, \tag{1.5}$$

L'ensemble des solutions $\mathbb{S}_1$ associé à $\mathcal{H}_1$ est représenté en noir sur la figure 1-2. L'ensemble de solutions vérifiant la contrainte $y_1 = x_1^2 + x_2^2$ correspond à l'anneau gris foncé et celui vérifiant la contrainte $y_2 = x_1 - x_2$ correspond à la bande gris clair. L'ensemble $\mathbb{S}_1$ est l'intersection de ces deux ensembles de solutions. Notre but tout au long de cette thèse va donc être de déterminer

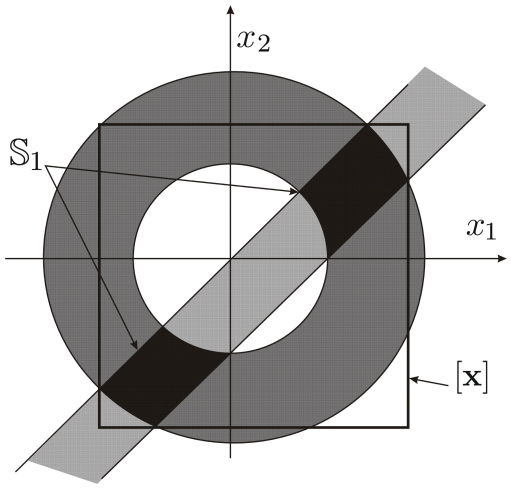

FIG. 1-2 – L'ensemble de solution $\mathbb{S}_1$ vérifiant le CSP $\mathcal{H}_1$ est en noir sur la figure.

le carré noir qui correspond à la plus petite boîte autour de l'ensemble de solutions $\mathbb{S}_1$. Bien sûr l'ensemble de solutions dans le cas de l'étalonnage ne sera pas caractérisé, mais une boîte garantie encadrant cet ensemble sera donnée.

1.4 Plan de la thèse

L'objectif de ce mémoire est de montrer l'évolution des méthodes de propagation de contraintes que nous avons utilisées et les façons dont elles ont été employées. Les échecs et les réussites obtenus sur le problème de l'étalonnage géométrique des robots séries nous ont guidés vers certaines solutions.

La thèse est donc formalisée de la façon suivante, le chapitre 2 présente l'application à l'étalon-

nage des robots séries mise sous la forme de CSP puis le chapitre 3 détaille les techniques par intervalles et les méthodes de propagation de contraintes.

Les deux chapitres suivants correspondent à l'implémentation des techniques de propagation de contraintes réalisée grâce à l'utilisation de graphes acycliques orientés (DAG). Le chapitre 4 présente les DAG et le chapitre 5 présente le logiciel IntervalPeeler, ainsi qu'un ensemble d'applications pouvant être résolues grâce à ce solveur. Les résultats obtenus pour l'étalonnage sont alors détaillés.

Le dernier chapitre 6 présente une amélioration pour IntervalPeeler permettant de mieux traiter des problèmes où les variables ont plusieurs occurences dans l'expression des contraintes.

Chapitre 2

L'étalonnage des robots

L'étalonnage des robots séries est le problème de grande dimension que nous avons traité tout au long de cette thèse de cette thèse. Les méthodes intervalles développées pour ce problème ne sont pas spécifiques et permettent le traitement d'autres problèmes mais c'est l'étalonnage des robots qui a eu le plus d'influence sur l'orientation de nos recherches. Ce travail a donné lieu à une publication [9]. D'autres travaux existent sur l'étalonnage avec les méthodes par intervalles, en particulier les travaux sur les robots parallèles de D. Daney et A. Neumaier (voir [20] et [55] pour plus de détails sur l'architecture des robots parallèles).

Les tâches réalisées par les robots demandent toujours plus de précision et les modèles mathématiques tentent d'apporter cette précision en contrôlant la position des robots dans un maximum de configurations. Lors de l'installation du robot les paramètres d'installation du robot dans son espace de travail ne sont pas précis de même avec l'usure des articulations les paramètres de construction du robot peuvent changer de valeur. Or pour positionner le robot avec précision le jeu de paramètres intervenant dans le modéledu robot est primordial. Ajuster ces paramètres avec l'étalonnage permet donc d'augmenter la précision.

Les robots séries, de par leur architecture, multiplient les erreurs tout au long de la chaîne cinématique et doivent donc pour éviter les erreurs de positionnement ajuster au plus près leur modèle géométrique. Les erreurs peuvent être dues à des facteurs géométriques (paramètres mal ajustés, axes non parallèles,...) mais aussi aux facteurs non pris en compte dans le modèle (élasticité, effets thermiques,...). Mais 95% des erreurs sont dues à des facteurs géométriques [39]. Pour traiter ces erreurs géométriques, un ajustement des valeurs des paramètres $\mathbf{p}$ du modèle peut être effectué, les erreurs non géométriques étant considérées comme des erreurs aléatoires bornées, mais dont nous ne connaissons pas forcément la densité de probabilité.

Le problème d'étalonnage revient donc à chercher les meilleures valeurs pour ces paramètres $\mathbf{p}$ tels que le modèle donne les coordonnées $\mathbf{x}$ les plus proches des coordonnées réelles du robot quelque soient les configurations $\mathbf{q}$ du robot.

Afin de visualiser au mieux les contraintes intervenant dans le problème, le chapitre est organisé

de la façon suivante. Les trois premières parties détaillent la génération du CSP avec ses trois composantes : les variables, les contraintes et les domaines. Pour détailler la liste des variables du problème, nous présenterons le problème de l'étalonnage, pour les contraintes nous détaillerons le modèle géométrique direct et pour les domaines, la procédure de simulation est présentée. La dernière partie traite des méthodes classiques utilisées pour l'étalonnage des robots (voir [33], [28], [65], [39], [14] et [43] pour des présentations de ces méthodes).

2.1 Génération des variables du CSP : problème de l'étalonnage

Les variables, présentes dans le CSP (nous renvoyons le lecteur à la page 13 pour la définition), sont les mêmes que celles utilisées pour la formalisation classique de l'étalonnage. L'ensemble des variables $\mathcal{D}$ est donc constitué des paramètres $\mathbf{p}$ du modèle, des mesures des coordonnées $\mathbf{x}$ de l'organe terminal, et des coordonnées articulaires $\mathbf{q}$ de chacune des configurations du robot. Mais pour bien comprendre à quoi correspondent ces variables, détailler le modèle du robot et le principe de l'étalonnage devient nécessaire.

2.1.1 Liaisons rotoïdes et prismatiques

Le modèle du robot reproduit l'ensemble des mouvements effectués par les articulations du robot réel. Il est donc composé d'articulations rotoïdes ou prismatiques et de corps liant ces articulations entre elles (voir figure 2-1). Chacune de ces articulations possède soit un axe de rotation pour les liaisons rotoïdes, soit un axe de translation pour les liaisons prismatiques. Ces axes définissent donc le mouvement qui sera repéré par les coordonnées articulaires $\mathbf{q}$ avec $q_i = \theta_i$ pour une rotation, $q_i = r_i$ pour une translation. Les variables contrôlant le mouvement du robot sont ainsi définies.

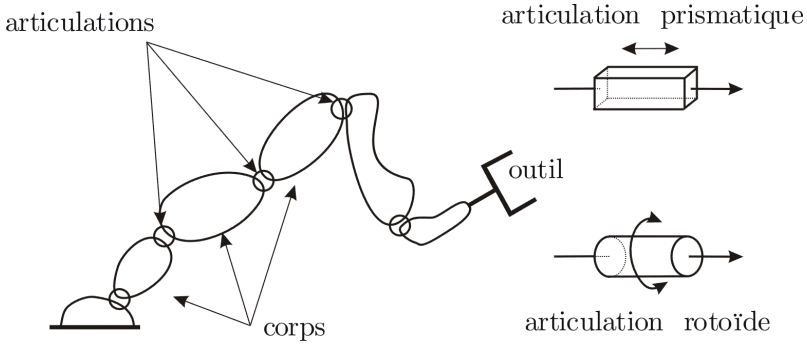

FIG. 2-1 – Schéma d'un robot et des liaisons rotoïdes et prismatiques.

Pour les corps, par exemple un corps qui relie l'articulation j à l'articulation $j+1$, la modélisation est effectuée par un ensemble de rotations et de translations fixes qui définissent les paramètres géométriques $\mathbf{p}$ du robot.

Enfin pour repérer l'organe terminal où se situe l'outil que porte le robot, les coordonnées $\mathbf{x}$ (position et orientation) sont introduites. Il ne reste alors plus qu'à introduire les différents modèles géométriques.

2.1.2 Modèles géométriques direct et inverse

Le modèle géométrique direct permet d'obtenir les coordonnées $\mathbf{x}$ (position et orientation) de l'organe terminal (l'outil) du robot en fonction des coordonnées articulaires $\mathbf{q}$

$$\mathbf{x} = \mathbf{mgd}(\mathbf{q}). \tag{2.1}$$

Le modèle géométrique inverse permet quant à lui d'obtenir les coordonnées articulaires $\mathbf{q}$ connaissant les coordonnées $\mathbf{x}$ de l'organe terminal sous certaines conditions (singularité, matrice inversible,...)

$$\mathbf{q} = \mathbf{mgi}(\mathbf{x}). \tag{2.2}$$

Ces deux modélisations vont permettre de calculer les coordonnées à partir des données fournies par le robot, les coordonnées non accessibles à la mesure seront ainsi calculées permettant de commander le robot.

Une commande avec génération de trajectoires dans l'espace des coordonnées articulaires $\mathbf{q}$ permet d'effectuer un déplacement articulation par articulation mais ne nécessite pas la connaissance des modèles géométriques des robots. En effet sur la figure 2-2 pour un déplacement d'un point initial $\mathbf{q}_i$ au point final $\mathbf{q}_f$, le contrôleur des moteurs des articulations permet à chaque étape $\mathbf{q}_t$ prévue par la génération de trajectoires d'effectuer le mouvement $\triangle\mathbf{q}$. Cette classe de mouvements reste toutefois limitée. Les utilisateurs de robots cherchent, la plupart du temps, à effectuer des déplacements dans l'espace de travail du robot.

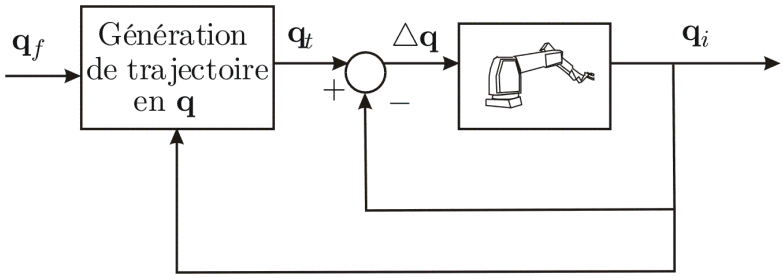

FIG. 2-2 – Schéma de contrôle du déplacement avec génération de la trajectoire dans l'espace articulaire $\mathbf{q}$.

C'est donc la génération de trajectoires dans l'espace des coordonnées $\mathbf{x}$ qui permet un déplacement plus proche de celui voulu par les utilisateurs. Dans ce cas, la transformation directe et inverse des coordonnées est nécessaire comme illustré sur la figure 2-3. Le modèle direct permet de connaître la position de l'organe terminal $\mathbf{x}_i$ connaissant la configuration articulaire du robot

$\mathbf{q}_i$. Avec cette position $\mathbf{x}_i$ et celle désirée $\mathbf{x}_f$, la génération de trajectoires peut alors donner une position intermédiaire $\mathbf{x}_t$ à atteindre que le modèle géométrique inverse convertit en coordonnées articulaires $\mathbf{q}_t$. Le robot peut ainsi être contrôlé en position dans le repère de l'utilisateur.

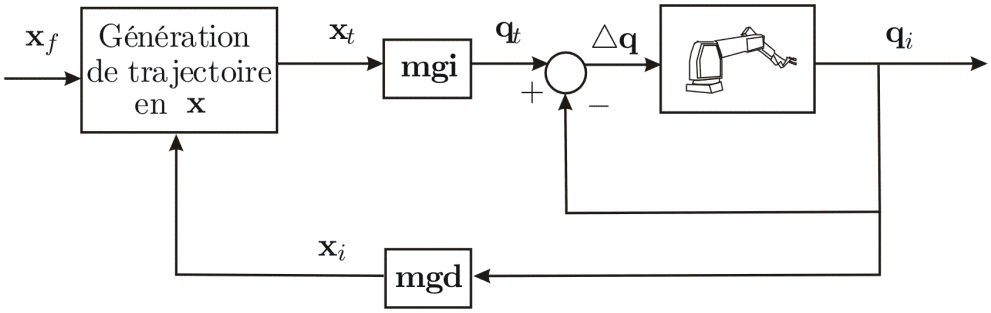

FIG. 2-3 – Schéma de contrôle du déplacement avec génération de la trajectoire dans l'espace articulaire $\mathbf{x}$.

Supposons maintenant l'existence d'erreurs sur les paramètres intervenants dans les deux modèles, cette erreur est introduite à chaque conversion de coordonnées entre $\mathbf{q}_i$ et $\mathbf{x}_i$ et entre $\mathbf{x}_t$ et $\mathbf{q}_t$. Le contrôle avec précision de la position du robot n'est donc plus assuré. Pour limiter ces erreurs, effectuer un étalonnage des paramètres du modèle permet d'ajuster les valeurs des paramètres aux valeurs réelles. Cette correction, bien qu'elle ne permet pas de corriger l'ensemble des erreurs (les erreurs de modèle par exemple), améliore grandement la précision.

2.1.3 Etalonnage géométrique

Réaliser l'étalonnage du robot revient à résoudre un problème d'estimation des paramètres géométriques $\mathbf{p}$ connaissant des mesures de position du robot dans différentes configurations. Pour l'étalonnage, les étapes suivantes sont donc réalisées :

1) Positionner le robot dans r configurations différentes $\mathbf{q}(1), ..., \mathbf{q}(r)$.

2) Mesurer la position et l'orientation de l'organe terminal $\mathbf{x}(1), ..., \mathbf{x}(r)$.

3) Résoudre le problème d'estimation pour déterminer $\mathbf{p}$ sachant qu'il vérifie le modèle géométrique direct

$$\mathbf{x}(k) = \mathbf{f}(\mathbf{p}, \mathbf{q}(k)),\ k \in \{1, ..., r\}. \tag{2.3}$$

Remarque 2.1.1 *La mesure de la position et de l'orientation avec des capteurs extérieurs est la méthode classiquement utilisée pour l'étalonnage. Pour notre problème, nous prendrons trois points fixés sur l'organe terminal du robot (*$\mathbf{x}(i)$ *devient un vecteur appartenant à* $\mathbb{R}^9$*), ces trois points permettant de retrouver l'orientation de l'outil. D'autres méthodes de mesures sont possibles, avec des mesures de distances et des contraintes géométriques où le robot est placé dans deux configurations articulaires différentes pour la même position ou orientation. (voir pour plus*

de détails le livre de E. Dombre et W. Khalil [44])

Remarque 2.1.2 *Dans notre cas l'étape 3) sera composée de deux étapes : 3.1) la génération des contraintes grâce au modèle géométrique direct et 3.2) la résolution du CSP par les méthodes intervalles.*

2.1.4 Variables du robot STAUBLI rx90

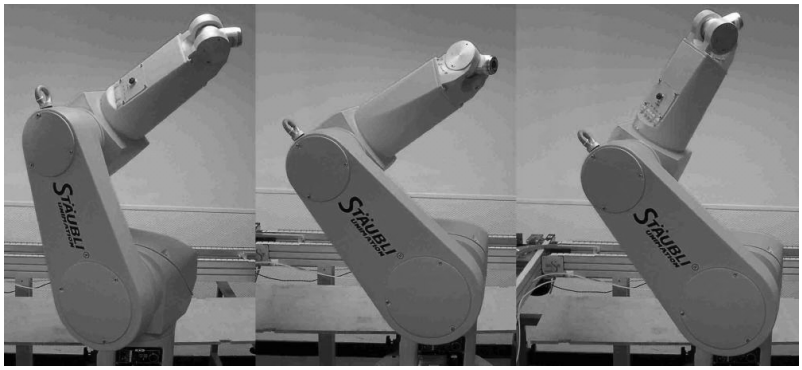

FIG. 2-4 – Robot Staubli RX90 dans différentes configurations.

Le robot Staubli RX90 de la figure 2-4 est un robot de type série possédant six degrés de liberté d'axes rotoïdes. Une configuration du robot est donnée par le vecteur des coordonnées articulaires

$$\mathbf{q} = (q_1, ..., q_6)^t \in \mathbb{R}^6, \tag{2.4}$$

où les q_i sont les angles des articulations du robot et sont exprimés en radians.

Le schéma de la figure 2-5 représente le robot dans quatre configurations. Elles correspondent aux vecteurs de configurations suivants :

$$\begin{array}{l} \mathbf{q}_a = (0,0,0,0,0,0)^t, \\ \mathbf{q}_b = \left(0,0,0,0,-\frac{\pi}{6},0\right)^t, \\ \mathbf{q}_c = \left(0,0,\frac{\pi}{6},0,-\frac{\pi}{6},0\right)^t, \\ \mathbf{q}_d = (0,-1.4,3.7,0,0.3,0)^t. \end{array} \tag{2.5}$$

La configuration $\mathbf{q}_a$ est la position de référence où l'ensemble des variables articulaires sont nulles. La rotation de l'avant-dernière articulation rotoïde permet d'obtenir la configuration $\mathbf{q}_b$. De la même façon, les deux autres configurations sont obtenues par rotation selon chaque axe des articulations rotoïdes. Le robot Staubli est conçu pour manipuler un *outil* ou *organe terminal*, qui se trouve en bout de la chaîne cinématique. L'outil sera repéré par trois points dont les coordonnées dans le repère de l'atelier forment le vecteur :

$$\mathbf{x} = \left(a_x^1, a_y^1, a_z^1, a_x^2, a_y^2, a_z^2, a_x^3, a_y^3, a_z^3\right)^t \in \mathbb{R}^9. \tag{2.6}$$

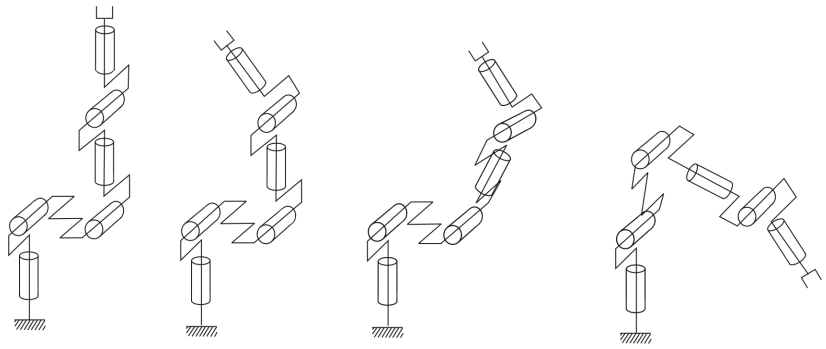

FIG. 2-5 – Schéma représentant quatre configurations pour les articulations du robot Staubli.

La structure du robot est définie par l'ensemble de paramètres géométriques qui représentent des paramètres de construction ou d'installation du robot (voir [44]). Les paramètres géométriques sont répartis de la façon suivante :

1) Les premiers paramètres du tableau de Denavit-Hartenberg (voir section 2.2) sont θ_0, r_0, α_1, d_1, θ_1^o et r_1, ils définissent la base du robot par rapport au repère atelier. Ces paramètres du robot sont mal connus, en effet le modèle géométrique fait reposer sur ces paramètres l'incertitude due à l'installation du robot dans le repère atelier.

2) Les paramètres b_x^i, b_y^i et b_z^i, $i = 1, 2, 3$ fixent trois points sur l'outil et permettent de définir un repère outil par rapport au repère du dernier corps. Trois contraintes relient ces paramètres car les distances entre les trois points sont connues.

3) Les paramètres géométriques α_j, d_j, θ_j^o et r_j sont liés au robot lui même et sont des données constructeur. Par exemple, d_3 correspond à la longueur du corps 3 du robot et r_4 à la longueur du corps 4. Les θ_j^o correspondent à l'offset sur la variables articulaires q_j, $j = 2, 3, 4, 5$.

Ces paramètres sont désignés par le vecteur :

$$\begin{aligned}\mathbf{p} = (&r_0, \alpha_1, d_1, r_1, \dots, \alpha_5, d_5, r_5, \alpha_6, d_6,\\ &\theta_0, \theta_1^o, \dots, \theta_5^o,\\ &b_x^1, b_y^1, b_z^1, b_x^2, b_y^2, b_z^2, b_x^3, b_y^3, b_z^3)^t.\end{aligned} \tag{2.7}$$

La liste des variables du CSP pour l'étalonnage du robot STAUBLI rx90, est donc définie par :

$$\mathcal{D} = \{\mathbf{q}(k), \mathbf{x}(k), \mathbf{p}\},\ k = \{1, ..., r\}, \tag{2.8}$$

avec les paramètres à estimer $\mathbf{p}$, les mesures sur l'organe terminal $\mathbf{x}(k)$ et les configurations articulaires associées $\mathbf{q}(k)$ pour r mesures sur le robot. Il reste, pour traiter le problème avec un CSP, à modéliser les contraintes et générer les domaines pour ces variables.

2.2 Génération des contraintes du CSP : modélisation des robots

Pour bien comprendre le problème et déterminer le comportement des algorithmes de propagation de contraintes sur les intervalles, la visualisation des contraintes devient nécessaire. Pour le problème de l'étalonnage, c'est le modèle géométrique direct qui correspond aux contraintes du CSP. Ce modèle de Denavit-Hartenberg modifié ([23] et [46]) est donc rappelé dans la partie 2.2.1. Cette modélisation classique est adaptée pour le calcul matriciel avec des réels mais ce formalisme n'est pas forcément le plus performant pour un travail avec des intervalles. Une approche non matricielle nécessitant une génération de contraintes un peu différente va donc être introduite dans la partie 2.2.2.

2.2.1 Modèle géométrique direct

Le modèle géométrique direct est construit à partir de la modélisation de Denavit-Hartenberg modifiés qui place un repère sur chaque articulation et permet ainsi, connaissant le vecteur des variables articulaires $\mathbf{q}$ et le vecteur des paramètres géométriques $\mathbf{p}$, de calculer la position d'un point de l'organe terminal dans le repère atelier. Le placement des repères s'effectue de la manière suivante :

1) placement de l'axe $\mathbf{z}_i$ dans l'axe de l'articulation i,

2) placement de l'axe $\mathbf{x}_i$ perpendiculaire à l'axe $\mathbf{z}_i$ perpendiculaire à l'axe $\mathbf{z}_{i-1}$. L'intersection entre $\mathbf{x}_i$ et $\mathbf{z}_i$ définit l'origine du repère O_i. Si $\mathbf{x}_i$ et $\mathbf{z}_i$ s'intersectent, l'origine est placée au point d'intersection.

Dans le cas particulier où l'axe $\mathbf{z}_i$ est parallèle à l'axe $\mathbf{z}_{i-1}$ alors le choix $\mathbf{x}_i$ n'est pas unique.

3) L'axe $\mathbf{y}_i$ est défini par le repère de sens direct $(O_i, \mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)$.

Une fois les repères placés sur le robot, l'utilisation des matrices de transformation pour la rotation pure et la translation pure permet d'établir la matrice de passage entre chaque repère :

$$^{i-1}\mathbf{T}_i = \mathbf{R}_{\mathbf{x}}(\alpha_i).\mathbf{T}_{\mathbf{x}}(d_i).\mathbf{R}_{\mathbf{z}}(\theta_i).\mathbf{T}_{\mathbf{z}}(r_i), \tag{2.9}$$

où les paramètres sont les suivants :
α_i : l'angle entre $\mathbf{z}_{i-1}$ et $\mathbf{z}_i$ selon $\mathbf{x}_{i-1}$
d_i : la distance séparant $\mathbf{z}_{i-1}$ de $\mathbf{z}_i$ le long de $\mathbf{x}_{i-1}$
θ_i : l'angle entre $\mathbf{x}_{i-1}$ et $\mathbf{x}_i$ selon $\mathbf{z}_i$
r_i : la distance séparant $\mathbf{x}_{i-1}$ de $\mathbf{x}_i$ le long de $\mathbf{z}_i$
Ce qui permet d'obtenir la matrice de transformation du repère de l'outil par rapport au repère de l'atelier :

$$^0\mathbf{T}_n = ^0\mathbf{T}_1.^1\mathbf{T}_2...^{n-2}\mathbf{T}_{n-1}.^{n-1}\mathbf{T}_n.$$

Et ainsi le modèle géométrique direct est appliqué aux trois points de l'outil pour obtenir la relation :

$$\mathbf{x} = \mathbf{f}(\mathbf{p}, \mathbf{q}), \tag{2.10}$$

où $\mathbf{f}$ s'exprime suivant l'algorithme :

Algorithme f

entrées : $\mathbf{q} = (q_1, ..., q_6)^t$,
$\mathbf{p} = \left(\alpha_j, d_j, r_j, \theta_0, \theta_j^o, b_x^i, b_y^i, b_z^i, ...\right)^t$.
sortie : $\mathbf{x} = \left(a_x^1, a_y^1, a_z^1, a_x^2, a_y^2, a_z^2, a_x^3, a_y^3, a_z^3\right)^t$.

1 $\mathbf{M} := \begin{pmatrix} \cos q_6 & -\sin q_6 & 0 & 0 \\ \sin q_6 & \cos q_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$; $\mathbf{M} := \begin{pmatrix} 1 & 0 & 0 & d_6 \\ 0 & \cos\alpha_6 & -\sin\alpha_6 & 0 \\ 0 & \sin\alpha_6 & \cos\alpha_6 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .\mathbf{M}$;

2 Pour $j := 5$ à 1,

3 $\theta := \theta_j^o + q_j$;

4 $\mathbf{M} := \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & r_j \\ 0 & 0 & 0 & 1 \end{pmatrix} .\mathbf{M}$; $\mathbf{M} := \begin{pmatrix} 1 & 0 & 0 & d_j \\ 0 & \cos\alpha_j & -\sin\alpha_j & 0 \\ 0 & \sin\alpha_j & \cos\alpha_j & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .\mathbf{M}$;

5 fin pour

6 $\mathbf{M} := \begin{pmatrix} \cos\theta_0 & -\sin\theta_0 & 0 & 0 \\ \sin\theta_0 & \cos\theta_0 & 0 & 0 \\ 0 & 0 & 1 & r_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} .\mathbf{M}$; $\mathbf{M} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} .\mathbf{M}$;

7 pour $i := 1$ à 3, $\mathbf{b}^i = \begin{pmatrix} b_x^i & b_y^i & b_z^i & 1 \end{pmatrix}^t$;

8 $\mathbf{x} := \begin{pmatrix} \mathbf{M} & 0 & 0 \\ 0 & \mathbf{M} & 0 \\ 0 & 0 & \mathbf{M} \end{pmatrix} \begin{pmatrix} \mathbf{b}^1 \\ \mathbf{b}^2 \\ \mathbf{b}^3 \end{pmatrix}$.

Après développement scalaire des relations matricielles de cet algorithme, nous pouvons exprimer $\mathbf{f}$ comme une composition de produits, de sommes, et de fonctions trigonométriques. Le modèle est donc fortement non linéaire.

En effet nous obtenons des contraintes scalaires de la forme :

$$\begin{aligned} t = \quad & \sin\alpha_2 \sin\theta_1 \sin\alpha_3 \sin\theta_3 + (\cos\theta_1 \cos\theta_2 - \cos\alpha_2 \sin\theta_2 \sin\theta_1) \cos\theta_3 \\ & -(\cos\alpha_2 \cos\theta_2 \sin\theta_1 + \cos\theta_1 \sin\theta_2) \cos\alpha_3 \sin\theta_3 \end{aligned} . \tag{2.11}$$

2.2.2 Formulation sans matrice

La méthode matricielle permet d'obtenir le modèle géométrique direct grâce à la matrice de transformation ${}^0\mathbf{T}_n$, les variables $\mathbf{q}$ et les paramètres $\mathbf{p}$ étant inclus dans la matrice de trans-

formation. Ces expressions matricielles donnent donc une fonction scalaire pour chacune des coordonnées scalaires x, y, z de position et d'orientation. Mais lors de l'utilisation des méthodes par intervalles, rien ne garantit que la meilleure contraction soit effectuée sur chacune des étapes de rotation et de translation.

Prenons la contrainte matricielle suivante :

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} \begin{pmatrix} i & j \\ k & l \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}, \tag{2.12}$$

elle correspond aux deux contraintes scalaires :

$$x = v\left(l\left(af + bh\right) + j\left(bg + ae\right)\right) + u\left(ibg + k\left(af + bh\right) + iae\right), \tag{2.13}$$

$$y = v\left(l\left(cf + dh\right) + j\left(dg + ce\right)\right) + u\left(idg + k\left(cf + dh\right) + ice\right). \tag{2.14}$$

Pour un calcul avec les réels, les deux expressions sont équivalentes mais lors de l'utilisation des méthodes intervalles, deux problèmes se posent qui seront détaillés dans les chapitres 3 et 6. Le premier problème est la multi-occurrence de certaines variables, par exemple a qui apparaît 4 fois dans la contrainte (2.13), ce qui engendre un pessimisme lors du calcul avec des intervalles (voir problème de dépendance chapitres 3). Le deuxième problème correspond au traitement des rotations et translations par deux fonctions scalaires dissociées. Sur l'expression (2.12), si les matrices correspondent à des matrices de rotation 2×2, le développement scalaire donne deux expressions où la notion de rotation disparaît avec les contraintes prises séparément. L'obtention des plus petits intervalles encadrant l'ensemble solution n'est alors plus garantie. Une possibilité est alors de ne pas utiliser le formalisme matriciel et de générer une deuxième formulation avec une contrainte pour la rotation et pour la translation. La translation, ne modifiant qu'une seule coordonnée, est alors simplement représentée par une contrainte addition.

Formulation utilisant la contrainte angle. Comme pour le formalisme précédent, nous établissons une relation entre le vecteur des variables articulaires $\mathbf{q}$, le vecteur des paramètres géométriques $\mathbf{p}$ et la position $\mathbf{x}$ de trois points de l'organe terminal dans le repère atelier.

La relation 2.10 peut se réécrire sous la forme :

$$\mathbf{x} = \mathbf{g}(\mathbf{p}, \mathbf{q}), \tag{2.15}$$

où $\mathbf{g}$ est donné par l'algorithme suivant :

Algorithme g

entrées : $\mathbf{q} = (q_1, ..., q_6)^t$,

$$\mathbf{p} = \begin{pmatrix} r_0, \alpha_1, d_1, r_1, \dots, \alpha_5, d_5, r_5, \alpha_6, d_6, \\ \theta_0, \theta_1^o, \dots, \theta_5^o \end{pmatrix}^t.$$

$\mathbf{x} = \left(a_x^1, a_y^1, a_z^1, a_x^2, a_y^2, a_z^2, a_x^3, a_y^3, a_z^3\right)^t$.

1 Pour $i := 1$ à 3,

2 $\begin{pmatrix} c_1 & c_2 & c_3 \end{pmatrix}^t := \begin{pmatrix} b_x^i & b_y^i & b_z^i \end{pmatrix}^t$;

3 $\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} := angle(q_6, \begin{pmatrix} c_1 \\ c_2 \end{pmatrix})$;

4 $c_1 := c_1 + d_6$;

5 $\begin{pmatrix} c_2 \\ c_3 \end{pmatrix} := angle(\alpha_6, \begin{pmatrix} c_2 \\ c_3 \end{pmatrix})$;

6 Pour $k := 5$ à 1,

7 $c_3 := c_3 + r_k$;

8 $\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} := angle(\theta_k^o + q_k, \begin{pmatrix} c_1 \\ c_2 \end{pmatrix})$;

9 $c_1 := c_1 + d_k$;

10 $\begin{pmatrix} c_2 \\ c_3 \end{pmatrix} := angle(\alpha_k, \begin{pmatrix} c_2 \\ c_3 \end{pmatrix})$;

11 finpour

12 $c_3 := c_3 + r_0$;

13 $\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} := angle(\theta_0, \begin{pmatrix} c_1 \\ c_2 \end{pmatrix})$;

14 $\mathbf{x}_i := \begin{pmatrix} c_1 & c_2 & c_3 \end{pmatrix}^t$;

15 finpour

16 $\mathbf{x} := \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}$.

La fonction $angle(\beta, \mathbf{c})$ génère un vecteur de sortie $\mathbf{c'} \in \mathbb{R}^2$ résultat d'une rotation de $\mathbf{c}$, dans le sens direct, d'un angle β. Cette fonction permet donc de traiter les deux cas de rotation selon les axes $\mathbf{x}_i$ et $\mathbf{z}_i$. Le modèle géométrique direct est donc utilisé pour chacun des trois points situés sur l'organe terminal pour l'étalonnage mais est uniquement formulé avec des translations et des rotations sans recours aux expressions matricielles.

2.3 Génération des domaines

2.3.1 Mesures sur un robot réel

Sur le robot réel, plusieurs types de mesures peuvent être réalisés : soit les mesures uniquement de la position (x, y, z) de l'organe terminal avec par exemple un système de trois câbles fixés en trois points de coordonnées connues dans le repère atelier ou soit des systèmes de mesures de la distance radiale [5]. Ces mesures permettront d'étalonner certains paramètres, d'autres restant non identifiables par cette méthode (voir [45]). La mesure de l'orientation en plus de la position permet d'étalonner le maximum de paramètres. Elle est réalisée avec des théodolites [26], des lasers [72], des caméras [6], des systèmes ultrasons [13]... D'autres types de mesures peuvent être utilisés pour l'étalonnage, nous ne les détaillerons pas ici, une liste de ces techniques peut être trouvée dans [44].

Dans notre cas, nous avons simulé une mesure réalisée à l'aide d'une caméra et de LEDs situées sur trois points de l'organe terminal. La position dans le repère atelier de la caméra étant connue, elle permet de situer avec précision les trois LEDs fixées sur l'outil et ainsi d'obtenir le vecteur des trois positions $\mathbf{x} = \left(a_x^1, a_y^1, a_z^1, a_x^2, a_y^2, a_z^2, a_x^3, a_y^3, a_z^3\right)^t \in \mathbb{R}^9$.

Un jeu de mesures $(\mathbf{q}(k), \mathbf{x}(k))$ est alors obtenu en plaçant le robot dans la configuration articulaire $\mathbf{q}(k)$ puis en mesurant $\mathbf{x}(k)$. Pour effectuer un étalonnage de qualité, les configurations prises par le robot doivent être suffisamment excitantes. Un jeu de mesures n'utilisant pas une des articulations ne permettra pas d'étalonner les paramètres. Dans les faits, la prise d'un nombre n suffisant de configurations aléatoires permet d'obtenir $6n$ équations, un nombre supérieur à 5 fois le nombre de paramètres permettant de réaliser l'étalonnage.

Nous avons donc pour chaque configuration du robot, un vecteur articulaire $\mathbf{q}(k)$ et un vecteur de position des trois points de l'organe terminal $\mathbf{x}(k)$ mesurés. L'incertitude e sur chacune de ces mesures permet d'affecter à chaque variable un intervalle auquel elle appartient. A chaque mesure est ajoutée et retranchée e pour obtenir les bornes supérieure et inférieure de l'intervalle comme dans les exemples suivants :

$$[q] = [q_m - e_q, q_m + e_q] \tag{2.16}$$

et

$$[x] = [x_m - e_x, x_m + e_x] \tag{2.17}$$

où e_x et e_q sont les incertitudes estimées sur les mesures x_m et q_m.

2.3.2 Conditions de simulations

L'étalonnage des paramètres géométriques n'a pas été réalisé sur des mesures réelles, une procédure de simulation a donc été utilisée pour générer un jeu de données. Les paramètres et la

façon dont a été éffectuée la simulation pour se rapprocher de la réalité sont exposés ci-dessous.

Nous avons choisi pour valeurs nominales des paramètres géométriques du robot STAUBLI RX90, les valeurs données en mètre dans la table suivante :

j	α_j	d_j	θ_j^o	r_j
0	-	-	$\frac{\pi}{2}$	0.5
1	0.1	0	0	0
2	$-\frac{\pi}{2}$	0	0	0
3	0	0.5	0	0
4	$\frac{\pi}{2}$	0	0	0.5
5	$-\frac{\pi}{2}$	0	0	0
6	$\frac{\pi}{2}$	0	-	-

Tableau 2.1-Valeurs nominales des paramètres géométriques recherchés.

Les trois points fixés sur l'organe terminal sont repérés par leurs coordonnées b_x^i, b_y^i et b_z^i dans le repère lié au corps terminal. Ces coordonnées sont des paramètres géométriques pour l'étalonnage et sont choisies avec les valeurs données dans la table suivante :

i	b_x^i	b_y^i	b_z^i
1	0.1	0.2	0.1
2	0.1	0.1	0.2
3	0.2	0.1	0.1

Tableau 2.2-Coordonnées nominales des trois points fixés sur l'organe terminal.

En prenant une configuration aléatoire du vecteur $\mathbf{q}$, nous effectuons une simulation du modèle direct qui nous donne les positions $\mathbf{x} \in \mathbb{R}^9$ des trois points de l'organe terminal. Pour cinquante configurations $\mathbf{q}$ différentes, nous avons ainsi obtenu par simulation un ensemble de cinquante vecteurs de position $\mathbf{x}$. Afin de prendre en compte les incertitudes de mesures présentes dans les mesures réelles, nous avons ajouté à chaque composante des vecteurs $\mathbf{q}$ et $\mathbf{x}$ un bruit aléatoire uniforme compris dans l'intervalle d'incertitude sur la mesure. Le bruit ajouté sur les mesures de position est de l'ordre de $10^{-4}m$ pour des mesures de l'ordre du mètre. Sur les angles des coordonnées articulaires, le bruit est de l'ordre de $10^{-5}rad$. Pour obtenir les bornes des intervalles pour chaque configuration $(\mathbf{x}, \mathbf{q})$ contenant les points de simulation, nous ajoutons à la mesure ponctuelle, plus ou moins l'erreur. Nous obtenons ainsi un intervalle pour chaque variable correspondant à une mesure proche de la réalité.

Exemple 2.3.1 *La variable articulaire q_1 est tirée de façon aléatoire dans son domaine de variation $[-\pi; \pi]$. Avec cette valeur, a_x^1 est calculée par simulation du modèle géométrique.*

Sur ces deux valeurs est ajouté un bruit aléatoire uniforme

$$\widetilde{a}_x^1 = a_x^1 + e_x, \tag{2.18}$$

où e_x *appartient à l'intervalle* $[-10^{-4}; 10^{-4}]$ *correspondant à l'incertitude de mesure sur les capteurs mesurant* $\mathbf{x}$*, et*

$$\widetilde{q}_1 = q_1 + e_q, \tag{2.19}$$

où e_q *appartient à l'intervalle* $[-10^{-5}; 10^{-5}]$ *correspondant au pas du codeur générant* $\mathbf{q}$**.**

Les intervalles pour a_x^1 *et* q_1

$$[a_x^1] = [\widetilde{a}_x^1 - 10^{-4}; \widetilde{a}_x^1 + 10^{-4}], \tag{2.20}$$

$$[q_1] = [\widetilde{q}_1 - 10^{-5}; \widetilde{q}_1 + 10^{-5}], \tag{2.21}$$

permettent de prendre en compte les incertitudes sur les valeurs supposées réelles $\widetilde{a}_x^1$ *et* $\widetilde{q}_1$*.*

De même, il faut déterminer un intervalle de recherche pour chaque paramètre que nous cherchons à étalonner. Ces domaines de recherche correspondent à la connaissance que nous avons sur les paramètres avant de réaliser l'étalonnage.

Exemple 2.3.2 *Les paramètres de la base et la position des trois points sur l'organe terminal sont mal connus, nous prenons donc un domaine de recherche de* $\pm 0.1m$ *sur les longueurs et de* $\pm 0.17rad$ *pour les angles. Pour les paramètres du robot lui-même nous disposons d'une meilleure information, la recherche s'effectue à* $\pm 0.01m$ *pour les longueurs et à* $\pm 0.035rad$ *pour les angles.*

2.4 Méthodes classiques

2.4.1 Un problème de minimisation

Pour l'étalonnage, il existe des méthodes classiques (voir [33], [28], [65], [39], [14] et [43]). Ces méthodes cherchent à minimiser les erreurs entre les valeurs mesurées pour l'étalonnage et les valeurs calculées avec le modèle géométrique. La fonction à minimiser est obtenue grâce aux relations suivantes.

Pour chaque mesure de la position de l'organe terminal et de la configuration des articulations, le modèle géométrique nous donne :

$$\mathbf{x}(r) = \mathbf{mgd}(\mathbf{p}, \mathbf{q}(r)), \tag{2.22}$$

d'où la relation générale :

$$\mathbf{x}_m = \mathbf{h}(\mathbf{p}, \mathbf{q}_m), \tag{2.23}$$

où $\mathbf{x}_m = (\mathbf{x}(1), ..., \mathbf{x}(r))^t$ et $\mathbf{q}_m = (\mathbf{q}(1), ..., \mathbf{q}(r))^t$. Le problème de l'étalonnage s'exprime alors de la façon suivante :

$$\min_{\mathbf{p}} ||\mathbf{x}_m - \mathbf{h}(\mathbf{p}, \mathbf{q}_m)||^2 = \min_{\mathbf{p}} j(\mathbf{x}_m, \mathbf{p}, \mathbf{q}_m). \tag{2.24}$$

La recherche du minimum global $\mathbf{p}^*$ minimisant la fonction j est alors effectuée. La figure 2-6 propose un exemple de fonction à minimiser, le minimum global est c_3, la fonction possédant deux autres minima locaux c_1 et c_2.

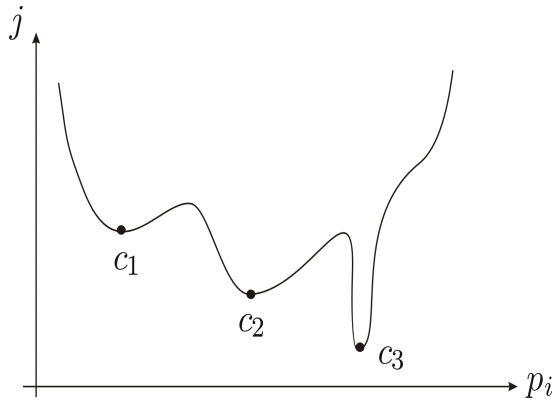

FIG. 2-6 – Illustration de la fonction j à minimiser sur l'axe p_i, les point c_1 et c_2 sont des minima locaux, c_3 étant le minimum global.

2.4.2 Les méthodes utilisées

Pour résoudre le problème de l'étalonnage, deux approches sont possibles : soit résoudre directement l'équation non linéaire (2.23), soit linéariser cette équation autour du point courant d'évaluation des paramètres puis résoudre le problème. Une des méthodes de descente utilisée sur le problème non linéarisé est la méthode de Levenberg Marquardt qui nécessite le calcul approché de la matrice hessienne à chaque itération. Pour le modèle linéarisé, le recours à la pseudo-inverse est utilisé pour obtenir à chaque itération les paramètres en fonction des mesures.

L'ensemble de ces méthodes, que ce soient les méthodes de descente basées sur le gradient [34] ou les méthodes par linéarisation, nécessite un point initial pour les paramètres $\mathbf{p}$. Ce point de départ pour les algorithmes ne doit pas être trop éloigné de la solution recherchée pour éviter que la méthode soit piégée dans un minimum local. Sur la courbe de gauche de la figure 2-7, les deux parcours fléchés représentent deux résolutions par une méthode de descente avec un point initial différent. Dans ces deux cas, la méthode est piégée dans des minima locaux.

Ces méthodes permettent donc l'obtention d'une estimée et d'une valeur d'incertitude. Mais la valeur donnée peut-être un minimum local et l'estimation de l'erreur faussée par l'approximation par la pente.

Dans les faits, la méthode donne de bons résultats. Le minimum global n'est peut être pas

obtenu, mais une validation du résultat avec de nouvelles mesures permet d'écarter les valeurs erronées. Le logiciel GECARO [44] permet de mettre en oeuvre ces méthodes sur des robots séries. D'autres méthodes pour la minimisation peuvent être employées comme des méthodes

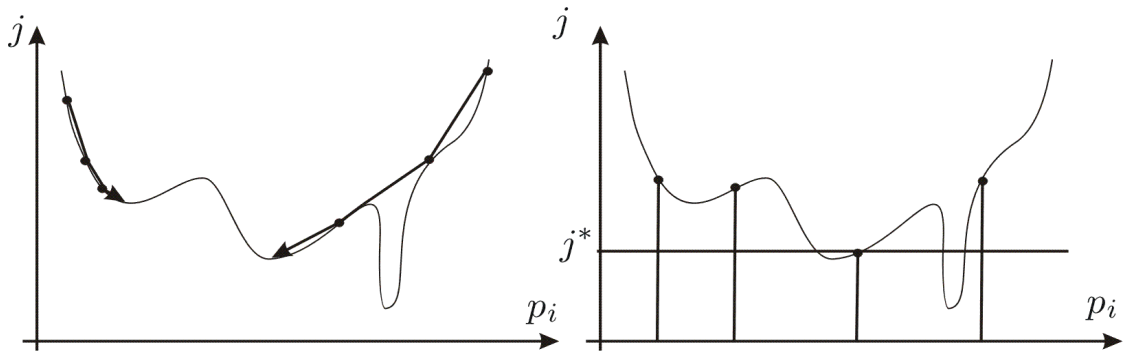

FIG. 2-7 – Illustration des méthodes de descente et des méthodes par tirage aléatoire.

basées sur un tirage aléatoire tel le recuit simulé ou monté carlo. Le tirage aléatoire permet d'explorer une plus large partie de l'espace en étant moins dépendant du point de départ de l'algorithme mais n'évite pas totalement la possibilité d'être coincé dans un minimum local. La grande dimension du problème induit un espace de recherche très grand. Le temps de résolution devient alors important pour permettre de couvrir l'ensemble de l'espace de recherche. Sur la figure 2-7, le schéma de droite représente le tirage de quatre valeurs aléatoires pour **p**, la valeur minimisant le critère est sélectionnée pour définir un voisinage où les prochains tirages aléatoires seront effectués. Le voisinage sera réduit à chaque itération permettant ainsi l'arrêt de l'algorithme sur la meilleure valeur minimisant le critère exploré par l'algorithme.

Dernièrement, des travaux ont été réalisés pour résoudre le problème de l'étalonnage avec des algorithmes génétiques. Cette méthode a été mise en oeuvre soit pour réaliser le choix des configurations où positionner le robot pour l'étalonnage [74], soit pour réaliser l'étalonnage lui même [77]. Le principe de ces algorithmes étant de ne plus faire un choix de recherche totalement aléatoire mais dirigé par des principes similaires à ceux existant dans la nature (population, mutation, reproduction et sélection). L'avantage de cette méthode est la possibilité d'adaptation qui rend possible une programmation sur le robot lui-même lors de son fonctionnement.

2.4.3 Avantages et inconvénients

Nous pouvons dégager des méthodes présentées précédemment trois conclusions : i) Elles permettent l'obtention d'un minimum local mais souvent de bonne qualité. ii) Une information sur l'incertitude est donnée mais cette valeur est seulement indicative puisque reposant sur le modèle linéarisé. iii) La prise en compte des erreurs de mesures est aussi possible sur ces méthodes, les données les plus aberrantes pouvant être écartées.

Pour chacune des méthodes proposées, la solution obtenue est ponctuelle, cette valeur correspond

au minimum d'un critère mais ne garantit pas la satisfaction des contraintes et la vérification de l'ensemble des mesures. Les équations du modèle géométrique peuvent alors ne pas être vérifiées pour certaines valeurs. Pour mieux visualiser le problème, prenons l'exemple d'une estimation à erreurs bornées suivant :

Exemple 2.4.1 *Si nous cherchons la meilleure droite passant par les quatre points définis par :*

$$x = [1, 2, 3, 4]$$

et :

$$y = [1, 2.5, 2, 2.5],$$

sachant que l'incertitude de mesure est de 0.5 *sur* y*. La méthode de régression linéaire nous donne la droite :*

$$y = 0.4x + 1,$$

alors que la droite :

$$y = 0.5x + 1,$$

est la seule droite permettant de vérifier les incertitudes. En effet sur la figure, la deuxième barre d'incertitude n'est pas atteinte par la droite obtenue avec la régression.

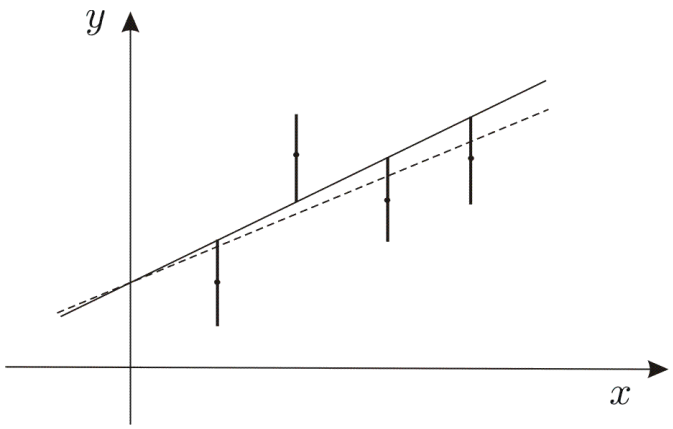

FIG. *2-8 – Tracé de la droite obtenue par régression linéaire (elle ne passe pas par la deuxième barre d'incertitude) et de la droite passant par l'ensemble des barres d'incertitude.*

2.5 Bilan

Dans ce chapitre nous avons proposé le problème de l'étalonnage géométrique des robots séries comme un problème de satisfaction de contraintes (CSP). Trouver l'ensemble de solution $\mathbb{S}_{csp}$ qui vérifie ce CSP peut être effectué par des méthodes de propagation de contraintes sur les intervalles. Les deux chapitres suivants détaillent la méthode de résolution proposée.

De nombreux autres problèmes se formalisent en CSP. L'estimation de paramètres, la détection de défauts, la commande robuste, l'estimation d'états sont des exemples de ces problèmes. Pour illustrer la généralisation des méthodes proposées dans ce manuscrit, certains de ces problèmes seront traités dans le chapitre 5.

Chapitre 3

Analyse par intervalles, propagation de contraintes

Dans ce chapitre sont abordées les bases des méthodes par intervalles utilisées au cours de cette étude. En effet, l'exemple d'étalonnage géométrique des robots a été un fil conducteur de la thèse pour l'utilisation de ces méthodes. Les outils et méthodes alors développés ont été orientés par cette application mais sont aussi valables pour tout autre problème qui se formalise sous forme de CSP.

Le problème de l'étalonnage est un problème de grande dimension, les techniques utilisant le découpage demandent alors un temps de résolution exponentiel. Pour cette raison nous avons choisi de n'utiliser que la propagation de contraintes sur les intervalles. Cette section présente donc le principe de cette méthode qui sera utilisée pour l'étalonnage du robot, les problèmes de ce type de résolution et les solutions envisagées (qui seront développés dans les chapitres suivants).

Ce chapitre est organisé autour de deux parties. La partie 3.1 détaille l'arithmétique et l'analyse par intervalles avec des méthodes telle que l'inversion ensembliste qui nous servira par la suite (pour illustrer la contraction obtenue pour des problèmes de faible dimension). Puis la partie 3.2 sur la propagation de contraintes présente les notions de contracteur, la projection de contraintes et un algorithme de propagation. Enfin, la partie 3.3 cloture ce chapitre en dressant un premier bilan.

3.1 Analyse par intervalles

3.1.1 Les précurseurs

Le calcul par intervalles est un concept utilisé de très nombreuses fois, oublié puis remis au goût du jour lors de la mise en place d'applications en particulier avec le calcul numérique. Les

premiers travaux de Young [76] ont mis en place l'analyse par intervalles sans que celle-ci soit tournée vers l'implémentation en machine.

En 1951 Paul S.Dwyer présente une première version [27] mais il faudra attendre les progrès de l'informatique pour que 3 mathématiciens présentent en 1966 au même moment des formalisations proches de l'implémentation en machine du calcul par intervalles : Mieczyslaw Warmus en Pologne [75], Teruo Sunaga au Japon [67] et Ramon E. Moore aux USA [57]. La version de Moore a été celle qui a le plus influencé la communauté du fait des solutions proposées pour l'implémentation et des publications successives de Moore.

Bien sûr, l'intérêt évident de l'analyse par intervalles est de donner une alternative de calcul non dépendante du codage en machine des réels. Le nombre de valeurs codées étant fini, calculer avec des valeurs ponctuelles présente une possibilité d'erreur qui est non négligeable en particulier pour certaines applications sensibles : suivi de trajectoires, ... L'analyse par intervalles fournit alors une information essentielle via la taille de l'intervalle sur l'erreur envisageable avec le calcul numérique standard. Le défi devient alors de trouver un intervalle de taille raisonnable pour que les résultats restent exploitables.

Depuis 1966 de nombreux travaux ont été réalisés (voir [32], [60] et [42]) avec en particulier des réussites en optimisation où les méthodes par intervalles, mises à part les méthodes formelles, restent les seules à permettre une optimisation globale en temps fini. Bien sûr, la dimension des problèmes traités reste un facteur limitant que toutes les recherches actuelles tentent de repousser.

3.1.2 Arithmétique par intervalles

Un intervalle $[x]$ est un ensemble fermé et connexe de $\mathbb{R}$. Il peut toujours s'écrire sous la forme

$$[x] = \{x \in \mathbb{R} \mid x^- \leq x \leq x^+, x^- \in \mathbb{R}, x^+ \in \mathbb{R}\} \tag{3.1}$$

où x^- et x^+ sont respectivement des bornes inférieures et supérieures de $[x]$.

La notation utilisée pour un intervalle est $[x^-; x^+]$. Un réel, par exemple 1, est alors représenté par un intervalle dégénéré avec les deux bornes égales, il est noté [1]. Nous pouvons aussi définir l'ensemble des intervalles de $\mathbb{R}$

$$\mathbb{IR} = \{A \subset \mathbb{R}, A = \{x \in \mathbb{R} \mid a \leq x \leq b, a \in \mathbb{R}, b \in \mathbb{R}\}\} \tag{3.2}$$

L'un des avantages de l'approche par intervalles réside dans la garantie donnée sur le contenu de l'intervalle. En effet, sur machine les réels ont une précision donnée selon le type de codage, le dernier chiffre étant donné par troncature ou par arrondi. Par exemple le nombre π avec une précision de deux chiffres après la virgule prend donc la valeur 3.14. Pour les intervalles, l'approche est différente le nombre π appartient à l'intervalle $[3,14; 3,15]$ et ce résultat est

garanti. Les outils utilisant l'arrondi extérieur commencent à être développés. En effet pour que les valeurs solutions restent inclues dans l'intervalle d'étude, un arrondi extérieur peut être mis en place. La borne inférieure est alors arrondie au premier réel inférieur codé en machine et la borne supérieure au réel supérieur.

Remarque 3.1.1 *Dans les programmes que nous avons implémentés, l'arrondi extérieur n'est pas utilisé. Nos programmes chercheront à montrer la faisabilité de l'approche intervalle. Dans ce rapport, nous ne nous intéresserons pas à ces problèmes que d'autres cherchent à résoudre. Néanmoins il est possible de mettre l'arrondi extérieur en place en suivant la norme IEEE [7].*

La taille d'un intervalle $[x]$ est définie par :

$$\omega([x]) = x^+ - x^-. \tag{3.3}$$

Son centre est défini par

$$c([x]) = \frac{x^+ + x^-}{2}. \tag{3.4}$$

Vecteur d'intervalles, produit cartésien d'intervalles, boîte, pavé

Un pavé ou vecteur d'intervalles de $\mathbb{IR}^n$ est un produit cartésien de plusieurs intervalles réels. Un pavé de $\mathbb{IR}^n$ est donc de la forme :

$$[\mathbf{x}] = \left[x_1^-; x_1^+\right] \times \left[x_2^-; x_2^+\right] \times ... \times \left[x_n^-; x_n^-\right] = \left[\mathbf{x}^-; \mathbf{x}^+\right] = \begin{pmatrix} \left[x_1^-; x_1^+\right] \\ \left[x_2^-; x_2^+\right] \\ .... \\ \left[x_n^-; x_n^+\right] \end{pmatrix} \tag{3.5}$$

C'est cette structure de pavé qui permet d'approximer des vecteurs incertains. Chaque composante du vecteur appartient à un intervalle.

Cependant la représentation par pavé implique une perte de précision. Un ensemble $\mathbb{S}$ de forme quelconque peut être encadré par un pavé $[\mathbf{x}]$. L'encadrement réalisé contient tous les points de $\mathbb{S}$ mais aussi tous les points du pavé n'appartenant pas à $\mathbb{S}$.

Sur la figure 3-1 le pavé qui encadre $\mathbb{S}$ contient aussi des points n'appartenant pas à $\mathbb{S}$ comme par exemple le couple (a_1, a_2). Toute la zone en gris clair est donc une incertitude sur la connaissance de $\mathbb{S}$ qui est due à la représentation des variables sous forme de pavés. Ce pessimisme engendré par les pavés est aussi appelé wrapping effect .

De la même manière que pour les intervalles, la taille du pavé est définie par la formule suivante :

$$\omega([\mathbf{x}]) = \max_i \left(\omega([x]_i)\right), \tag{3.6}$$

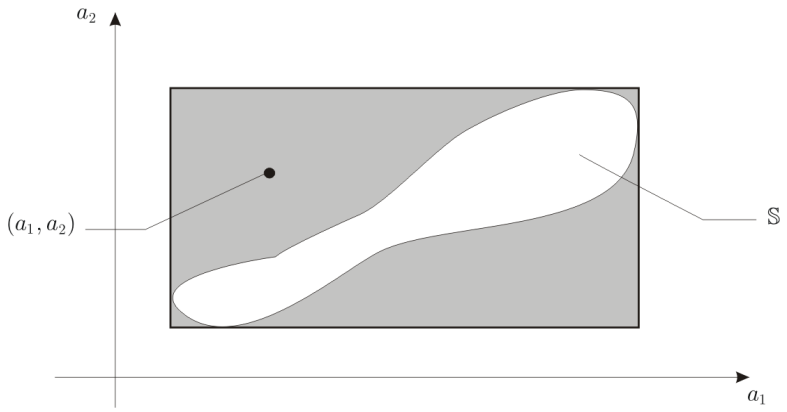

FIG. 3-1 – Illustration de l'encadrement d'un ensemble par une boîte, tous les points gris n'appartiennent pas à l'ensemble.

et le volume par

$$v([\mathbf{x}]) = \prod_i (\omega([x]_i)) \,. \tag{3.7}$$

Opérations algébriques sur les intervalles

Un ensemble d'opérations élémentaires peut être effectué sur les intervalles : c'est la généralisation des opérations élémentaires sur les réels.

Une variable incertaine x appartenant au domaine $[x]$ prend donc des valeurs qui sont contenues dans cet intervalle. Par exemple une variable x peut prendre n'importe quelle valeur dans le domaine $[x] = [-1; 3]$.

Additionner deux variables appartenant à deux intervalles différents revient à chercher l'intervalle contenant toutes les valeurs des additions de ces variables. Si $[y] = [0; 2]$ est le domaine pour la variable y, l'addition de x et y donne donc l'intervalle $[-1; 5]$.

Cet intervalle est déterminé par la somme des bornes inférieures qui donne -1 et celle des bornes supérieures qui donne 5.

L'ensemble des opérations est calculable de la même façon et les règles associées sont les suivantes :

$$\begin{aligned} [x] + [y] &= \{x + y \mid x \in [x], y \in [y]\} = [x^- + y^-; x^+ + y^+] \\ -[y] &= \{-y \mid y \in [y]\} = [-y^+; -y^-] \\ [x] - [y] &= \{x - y \mid x \in [x], y \in [y]\} = [x] + (-[y]) \end{aligned} \tag{3.8}$$

$$\begin{aligned}
[x] * [y] &= \{x * y \mid x \in [x], y \in [y]\} \\
&= [\ \min(x^- * y^-, x^- * y^+, x^+ * y^-, x^+ * y^+)\ ; \\
&\qquad \max(x^- * y^-, x^- * y^+, x^+ * y^-, x^+ * y^+)\ ] \qquad (3.9) \\
\frac{1}{[y]} &= \{\frac{1}{y} \mid y \in [y]\} = [\frac{1}{y^+}; \frac{1}{y^-}] \text{ si } 0 \notin [y], \ ]-\infty; +\infty[\text{ sinon.}
\end{aligned}$$

Une généralisation de ces règles est possible avec l'ajout des bornes infinies. En effet les intervalles permettent de prendre en compte les valeurs infinies. La division par un intervalle contenant zéro en particulier.

$$\begin{aligned}
\frac{1}{[y]} &= \emptyset \text{ si } [y] = [0,0] \\
&= [\frac{1}{y^+}; \frac{1}{y^-}] \text{ si } 0 \notin [y] \\
&= [\frac{1}{y^+}; \infty[\text{ si } y^- = 0 \text{ et } y^+ > 0 \qquad (3.10) \\
&=]-\infty; \frac{1}{y^-}] \text{ si } y^- < 0 \text{ et } y^+ = 0 \\
&=]-\infty; +\infty[\text{ si } y^- < 0 \text{ et } y^+ > 0 \\
\frac{[x]}{[y]} &= [x] * \left(\frac{1}{[y]}\right) \qquad (3.11)
\end{aligned}$$

Remarque 3.1.2 *L'intérêt de l'ajout de bornes $-\infty$ et $+\infty$ est l'obtention d'un résultat quelque soient les singularités et les points où les fonctions ne sont pas définies et la possibilité d'explorer $\mathbb{R}$ dans son intégralité pour les recherches de preuves. Dans les faits, si nous obtenons le résultat $]-\infty; +\infty[$, une étude reste nécessaire et ce résultat n'a que peu de valeur.*

Opérations ensemblistes sur les intervalles

L'intersection et l'union d'ensembles sont deux notions très utilisées. Elles sont définies comme : pour l'intersection

$$[x] \cap [y] = \{x \in [x] \text{ et } y \in [y]\}, \qquad (3.12)$$

et pour l'union

$$[x] \cup [y] = \{x \in [x] \text{ ou } y \in [y]\}. \qquad (3.13)$$

L'union de deux intervalles ne donne pas toujours un intervalle. Par exemple l'union des intervalles $[1;2]$ et $[3;4]$, donne pour résultat un ensemble disjoint de deux intervalles. Bien que ce résultat soit intéressant, il rend difficile la manipulation de l'union, le nombre d'intervalles disjoints lors de la résolution des problèmes pouvant croître rapidement. Il faut donc introduire une notion d'union différente qui renvoie le plus petit intervalle contenant l'ensemble des solutions de l'union. L'intervalle solution est donc l'encadrement extérieur de l'union définie par la formule

$$[x] \sqcup [y] = [[x] \cup [y]] \qquad (3.14)$$

où l'opérateur [...] renvoie le plus petit intervalle encadrant l'union.

Ces opérations sont codées par les formules suivantes pour des intervalles non vides :

$$\begin{aligned} [x] \cap [y] &= \emptyset \quad \text{si } (x^+ < y^-) \text{ ou } (y^+ < x^-) && (3.15)\\ &= [\max(x^-, y^-); \min(x^+, y^+)] \text{ sinon} && (3.16)\\ [x] \sqcup [y] &= [\min(x^-, y^-); \max(x^+, y^+)] && (3.17) \end{aligned}$$

Le problème de dépendance

Pour illustrer une dernière fois le problème de dépendance, prenons l'exemple de l'évaluation par intervalles de deux expressions quand $[x] = [-1; 1]$:

$$\begin{aligned} [x] + [x] - [x] &= [-1; 1] + [-1; 1] - [-1; 1] = [-3; 3] && (3.18)\\ [x] &= [-1; 1] \end{aligned}$$

L'intervalle solution est dépendant du nombre d'occurrences de la variable x.

$$\begin{aligned} [x] + [x] - [x] &= \{a + b - c \mid a \in [x], b \in [x], c \in [x]\} && (3.19)\\ [x] &= \{a \in [x]\} && (3.20) \end{aligned}$$

L'expression 3.19 correspond donc à l'ensemble des réels $a+b-c$ pour tout a, b et c pris chacun séparément dans l'intervalle $[-1; 1]$, et non pas à l'ensemble des réels avec $a = b = c$.

Du fait de ce problème de dépendance, il faut veiller à limiter le nombre d'occurrences de chaque variable afin d'obtenir des intervalles plus petits.

3.1.3 Les principes de base

L'ensemble défini par

$$\mathbf{f}([\mathbf{x}]) = \{\mathbf{f}(\mathbf{x}) | \mathbf{x} \in [\mathbf{x}]\} \tag{3.21}$$

correspond à l'ensemble des valeurs que peut prendre la fonction $\mathbf{f}$ quand les variables appartiennent aux domaines intervalles $[\mathbf{x}]$. Trouver des techniques pour manipuler ces ensembles quelque soit la fonction $\mathbf{f}$ est essentiel pour traiter les problèmes et évaluer ces fonctions avec des intervalles.

Fonction d'inclusion

Définition 3.1.1 *Soit* $\mathbf{f}$ *une fonction de* $\mathbb{R}^n$ *à valeurs dans* $\mathbb{R}^m$*. La fonction intervalles* $[\mathbf{f}] : \mathbb{R}^n \to \mathbb{R}^m$ *est une fonction d'inclusion pour* $\mathbf{f}$ *si*

$$\forall \mathbf{x} \in \mathbb{R}^n, \ \mathbf{f}([x]) \subset [\mathbf{f}]([x]) \tag{3.22}$$

et si $\mathbf{f}([x]) = [\mathbf{f}]([x])$ *alors la fonction d'inclusion est minimale.*

Exemple 3.1.1 *Pour la fonction $f : x \rightarrow sin(x)$ une fonction d'inclusion triviale est définie de la façon suivante*

$$\forall [x] \in \mathbb{IR},\ [\sin]([x]) = [-1; 1], \tag{3.23}$$

cette fonction n'est pas minimale sauf si $x \in [-\pi; \pi]$ mais permet d'avoir une inclusion pour le domaine de la fonction sin. Une fonction plus intéressante prend en compte la monotonie de la fonction sinus.

Une fonction d'inclusion $[\mathbf{f}]$ pour $\mathbf{f}$ est *convergente*, si pour toute suite $[\mathbf{x}](k)$,

$$\lim_{k \rightarrow +\infty} \omega([\mathbf{x}](k)) = 0 \Rightarrow \lim_{k \rightarrow +\infty} \omega([\mathbf{f}]([\mathbf{x}](k))) = 0. \tag{3.24}$$

C'est cette propriété de convergence qui est utilisée pour démontrer que les algorithmes ensemblistes convergent.

Pour les fonctions $\sqrt{}$, sin, cos, tan, exp,... la décomposition en domaines d'étude où les fonctions sont monotones permet d'obtenir directement la fonction d'évaluation minimale.

Exemple 3.1.2 *La fonction carré est divisée en deux parties, $[x] \cap]-\infty; 0]$ où f est décroissante avec $[f]([x] \cap]-\infty; 0]) = [(x^+)^2; (x^-)^2]$ et $[x] \cap]0; +\infty]$ où f est croissante avec $[f]([x] \cap]0; +\infty]) = [(x^-)^2; (x^+)^2]$. La fonction finale est alors donnée par l'union de ces deux intervalles.*

Fonction d'évaluation

La *fonction d'évaluation naturelle* est obtenue par le remplacement de chacune des variables par leur domaine intervalle associé, l'analyse par intervalles permettant de réaliser le calcul. Cette évaluation permet d'obtenir un intervalle réalisant une fonction d'inclusion mais ne garantit pas l'obtention de l'intervalle minimal encadrant la fonction. En effet, si une variable apparaît plusieurs fois dans l'expression de la fonction, ces multi-occurrences engendrent éventuellement un pessimisme dans l'évaluation naturelle de la fonction.

Exemple 3.1.3 *Soit la fonction suivante :*

$$f_1(x) = 2.x.x - 2.x - 1, \tag{3.25}$$

représentée sur la figure 3-2. Son évaluation naturelle pour les intervalles $[0; 1]$, $[1; 2]$, et $[2; 3]$ donne respectivement les intervalles $[-3; 1]$, $[-3; 5]$ et $[1; 13]$. La première idée pour diminuer ce pessimisme est de réduire le nombre d'occurrences de x en utilisant la fonction carré

$$f_2(x) = 2.x^2 - 2.x - 1. \tag{3.26}$$

Mais l'évaluation naturelle de cette fonction ne donne pas plus de précision tout du moins sur ces domaines. C'est le recours à une transformation formelle (forme de Horner)

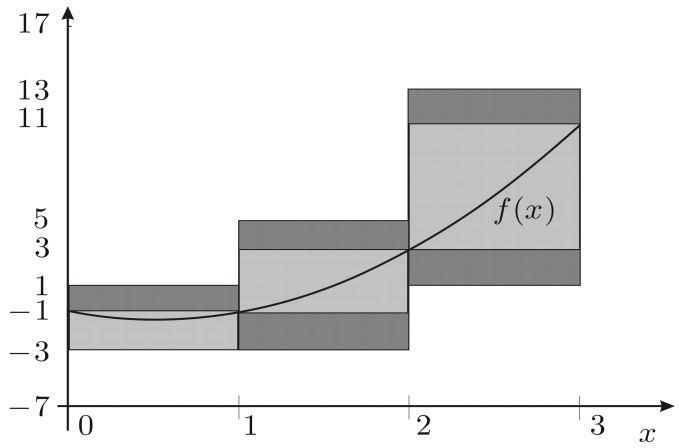

FIG. *3-2 – Illustration de la fonction d'évaluation naturelle pour f_1 en gris foncé et f_2 en gris clair.*

$$f_3(x) = x.(2.x - 2) - 1, \tag{3.27}$$

qui permet d'obtenir des intervalles plus petits respectivement $[-3; -1]$, $[-3; 5]$ *et* $[3; 11]$, *sans toutefois que cette fonction ait avec l'évaluation naturelle une fonction d'inclusion minimale. La fonction d'évaluation minimale est obtenue pour :*

$$f_4(x) = 2.(x - \frac{1}{2})^2 - \frac{3}{2}, \tag{3.28}$$

où x n'apparaît qu'une seule fois.

Remarque 3.1.3 *En règle générale, la diminution du nombre d'occurrences des variables dans l'expression de f permet de réduire le pessimisme (voir [66] pour les formes de Horner et Bernstein). Une manipulation formelle permettant de factoriser dans le but de réduire le nombre d'occurrences des variables peut améliorer les évaluations.*

D'autres fonctions d'évaluation permettent d'obtenir une meilleure évaluation des fonctions multi-occurrentes en particulier pour les petits pavés. Soit $f : \mathbb{R}^n \to \mathbb{R}$ une fonction dérivable sur $[\mathbf{x}]$ et $\mathbf{x}_0$ le centre de ce pavé $[\mathbf{x}]$. Le théorème de la valeur moyenne s'énonce de la façon suivante

$$\forall \mathbf{x} \in [\mathbf{x}], \exists \mathbf{c} \in [\mathbf{x}], f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{c}).(\mathbf{x} - \mathbf{x}_0). \tag{3.29}$$

Par conséquent

$$f(\mathbf{x}) \in f(\mathbf{x}_0) + \nabla f([\mathbf{x}]).(\mathbf{x} - \mathbf{x}_0), \tag{3.30}$$

et l'ensemble des points

$$f([\mathbf{x}]) \subset f(\mathbf{x}_0) + [\nabla f]([\mathbf{x}]).([\mathbf{x}] - \mathbf{x}_0), \tag{3.31}$$

ce qui défini la *forme centrée standard*

$$[f]([\mathbf{x}]) \triangleq f(\mathbf{x}_0) + [\nabla f]([\mathbf{x}]).([\mathbf{x}] - \mathbf{x}_0), \tag{3.32}$$

où $[\nabla f]$ est la fonction d'inclusion naturelle de la dérivée.

Exemple 3.1.4 *Sur l'exemple 3.1.3, la fonction* $f_2(x) = 2.x^2 - 2.x - 1$ *a pour évaluation naturelle sur l'intervalle* $[0; 1]$

$$[f]([0; 1]) = 2.[0; 1]^2 - 2.[0; 1] - 1 = [-3; 1].$$

Calculons maintenant l'évaluation grâce à la forme centrée. La dérivée de la fonction est

$$\frac{df_2}{dx}(x) = 4.x - 2,$$

son évaluation naturelle sur l'intervalle $[0; 1]$ *est*

$$[\frac{df_2}{dx}]([0; 1]) = 4.[0; 1] - 2 = [-2; 2].$$

Le centre de l'intervalle $[0, 1]$ *est* $\frac{1}{2}$ *nous avons donc la fonction d'inclusion*

$$\begin{aligned} [f_2]([0; 1]) &= f_2(\frac{1}{2}) + [\frac{df_2}{dx}]([0; 1]).([0; 1] - \frac{1}{2}) \\ &= -\frac{3}{2} + [-2; 2].([0; 1] - \frac{1}{2}) = [-\frac{5}{2}; -\frac{1}{2}]. \end{aligned}$$

Grâce à cette évaluation nous obtenons une meilleure estimation de la fonction évaluée sur l'intervalle $[0; 1]$.

Remarque 3.1.4 *La forme centrée peut aussi être utilisée avec le développement de Taylor à l'ordre 2*

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0).(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^t \mathbf{H}_f(\mathbf{d}).(\mathbf{x} - \mathbf{x}_0), \tag{3.33}$$

où $\mathbf{H}_f$ *est la matrice Hessienne de* f.

Test d'inclusion

Soit le vecteur $\mathbf{x} \in \mathbb{R}^n$ appartenant au pavé $[\mathbf{x}]$. Pour comparer ce pavé avec un autre pavé $[\mathbf{y}] \in \mathbb{IR}^n$, le recours à un test d'inclusion est nécessaire et permet de traiter les résultats que nous donne l'analyse par intervalles.

Si $[\mathbf{x}]$ est inclus dans $[\mathbf{y}]$, le test est vrai.

S'il existe i tel que $[x_i]$ et $[y_i]$ sont disjoints, le test est faux.

S'il existe i tel que $x_i^- < y_i^- < x_i^+$ ou $x_i^- < y_i^+ < x_i^+$, le test est indéterminé car $[x]$ contient à la fois des valeurs appartenant et n'appartenant pas à $[y]$.

Inversion ensembliste

Lorsque le problème est le suivant :

$$\mathbf{y} = \mathbf{f}(\mathbf{x}), \tag{3.34}$$

avec pour les variables les domaines :

$$\mathbf{x} \in [\mathbf{x}] \text{ et } \mathbf{y} \in [\mathbf{y}], \tag{3.35}$$

la caractérisation de l'ensemble :

$$\mathbb{S} = \{\mathbf{x} \in [\mathbf{x}] \mid \mathbf{f}(\mathbf{x}) \in [\mathbf{y}]\}, \tag{3.36}$$

est possible avec l'analyse par intervalles.

Pour caractériser l'ensemble $\mathbb{S}$, un sous pavage de $[\mathbf{x}]$ est réalisé grâce à l'algorithme SIVIA (Set Inversion via Interval Analysis) [38, 59],

Algorithme SIVIA$([\mathbf{x}], [\mathbf{y}])$	
1	$[\mathbf{y}_1] := [\mathbf{f}]([\mathbf{x}])$;
2	si $([\mathbf{y}_1] \subset [\mathbf{y}])$ $[\mathbf{x}]$ est solution
3	sinon
4	si ($[\mathbf{y}_1]$ et $[\mathbf{y}]$ disjoint)
5	$[\mathbf{x}]$ n'est pas solution
6	sinon si $(w([\mathbf{x}]) > \epsilon)$
7	**bisect**$([\mathbf{x}], [\mathbf{x}_1], [\mathbf{x}_2])$
8	**SIVIA**$([\mathbf{x}_1], [\mathbf{y}])$;
9	**SIVIA**$([\mathbf{x}_2], [\mathbf{y}])$.

L'évaluation $[\mathbf{f}]([\mathbf{x}])$ est comparée avec $[\mathbf{y}]$ grâce au test d'inclusion, si $[\mathbf{f}]([\mathbf{x}])$ est inclue dans $[\mathbf{y}]$, le pavé courant $[\mathbf{x}]$ est alors entièrement solution et appartient à la partie du sous pavage caractérisant l'approximation intérieure de $\mathbb{S}$. Si le test est négatif, $[\mathbf{x}]$ n'appartient pas à $\mathbb{S}$ et fait partie de l'approximation extérieure.

Dans le cas indéterminé du test, le pavé courant contient à la fois des points solutions et non solutions, l'algorithme est alors relancé en découpant $[\mathbf{x}]$ en deux, puis en évaluant l'image par $\mathbf{f}$ des boites $[\mathbf{x}_1]$ et $[\mathbf{x}_2]$. La condition d'arrêt de cet algorithme récursif étant l'obtention d'une taille $\omega([\mathbf{x}])$ inférieure à une valeur ϵ donnée par l'utilisateur. La figure 3-3 illustre la comparaison entre la fonction d'évaluation $[\mathbf{f}]([\mathbf{x}])$ et $[\mathbf{y}]$ lors du test d'inclusion.

Ces algorithmes peuvent résoudre des problèmes de dimension de l'ordre de 5, cette limite étant liée au caractère exponentiel de ces algorithmes. Dans le cas de dimensions supérieures, la résolution est possible uniquement pour des problèmes spécifiques (petits domaines de solutions, solutions quasi ponctuelles isolées,...). L'apport de méthodes spécifiques comme la projection d'ensemble avec des techniques de découpage [17] et [21] permet d'augmenter la dimension des

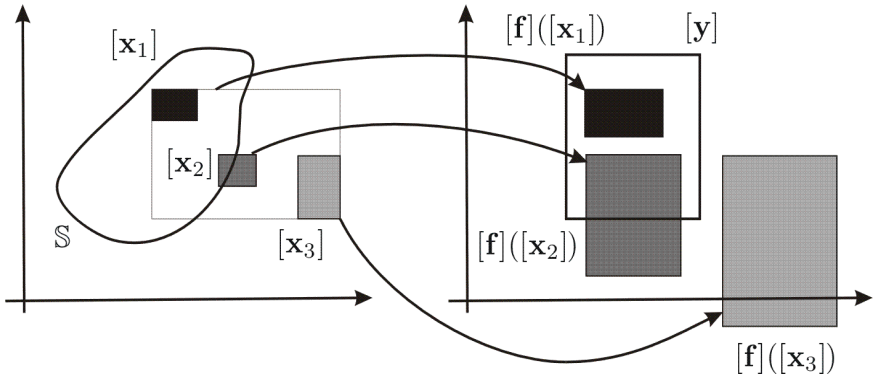

FIG. 3-3 – Schéma représentant l'inversion ensembliste via analyse par intervalles.

problèmes traités. L'algorithme se concentre sur les dimensions projetées et caractérise l'ensemble sur ces dimensions.

Dans tous les cas, les problèmes de grande dimension ne permettent pas à l'heure actuelle une caractérisation complète de l'ensemble de solutions. Nous nous sommes donc orientés vers l'obtention d'une caractérisation uniquement partielle (boîte encadrant l'ensemble de solutions, solution garantissant l'appartenance à $\mathbb{S}$,...).

La présentation de la méthode d'inversion ensembliste permet donc de voir où se situe le travail de cette thèse. La caractérisation de l'ensemble des solutions de façon complète sera utilisée uniquement pour comparer la caractérisation partielle par une boîte extérieure avec l'ensemble de solutions sur des cas d'étalonnage de plus faible dimension. (voir l'étalonnage du robot SCARA dans le chapitre 4).

L'analyse par intervalles permet donc la caractérisation d'un ensemble par inversion ensembliste. D'autres problèmes sont traités via les intervalles comme l'image directe d'un ensemble par une fonction, les problèmes d'optimisation, les démonstrations de preuves et enfin de nombreuses méthodes basées sur les réels ont été transposées aux intervalles, méthode de Newton par intervalles, Méthode de Gauss,... elles peuvent être trouvées dans [60], [32] et [51],mais ne seront pas abordées dans ce rapport.

3.2 L'apport de la communauté contraintes entière et logique

L'ensemble des méthodes par intervalles ont été développées dans l'optique d'évaluer des fonctions. Avec les fonctions $\mathbf{f}$, l'analyse par intervalles permet d'obtenir un intervalle encadrant cette fonction $[\mathbf{f}]([\mathbf{x}])$. Si nous possédons une information sur $[\mathbf{y}]$ quand $\mathbf{y} = \mathbf{f}(\mathbf{x})$ alors les algorithmes d'inversion ensembliste s'appliquent pour obtenir la caractérisation de l'ensemble :

$$\mathbb{S} = \{\mathbf{x} \in [\mathbf{x}] \mid \mathbf{f}(\mathbf{x}) \in [\mathbf{y}]\}. \tag{3.37}$$

Mais ces fonctions peuvent aussi se voir comme un ensemble de *contraintes* entre les variables x_i et y_j. Or des méthodes de résolution par contraintes existent quand les variables sont entières [53]. C'est Cleary [19] et Davis [22] en 1987 qui pour la première fois transposent ces méthodes aux domaines continus. Depuis, de nombreuses améliorations ont été apportées aux algorithmes de propagation de contraintes du type de celui de Waltz [73], nous pouvons également citer [30], [11], [71], [35] et [12].

De nombreux solveurs existent tels que NUMERICA [69], ILOGSOLVER [62], INTLAB [2], CLP(BNR) [12], PROLOG IV [61], REALPAVER [31], ou de librairies ALIAS [56].

Dans cette partie, les définitions et notions utiles (parties 3.2.2, 3.2.1 et 3.2.3), vont être illustrées pour permettre de présenter le fonctionnement de la propagation de contraintes (partie 3.2.5). La partie 3.2.4 détaille les opérateurs de projection des contraintes primitives qui sont les briques élémentaires et essentielles des algorithmes de propagation. Après avoir fait le bilan et présenté quelques problèmes (partie 3.2.6), nous présenterons une amélioration des contraintes primitives avec les contraintes globales (partie 3.2.7).

3.2.1 Notion de projection

Prenons un ensemble de contraintes

$$\begin{aligned} c_1 &: f_1(\mathbf{x}) = 0, \\ c_2 &: f_2(\mathbf{x}) = 0, \\ &\vdots \\ c_m &: f_m(\mathbf{x}) = 0. \end{aligned} \tag{3.38}$$

Quand $\mathbf{x}$ vérifie une de ces contraintes, il appartient à l'ensemble

$$\mathbb{S}_j = \{\mathbf{x} \in \mathbb{R}^n \mid c_j : f_j(\mathbf{x}) = 0\}. \tag{3.39}$$

Prenons maintenant une variable x_i, l'ensemble des valeurs x_i vérifiant la contrainte j définissent un ensemble de solution, une projection de l'ensemble $\mathbb{S}_j$ sur l'axe x_i. La *projection* sur x_i de $\mathbb{S}_j$ notée $\pi_i(\mathbb{S}_j)$ de l'ensemble $\mathbb{S}_j$ sur les domaines $[\mathbf{x}]$ ici $\mathbb{R}^n$ est définie comme

$$\pi_i(\mathbb{S}_j) = \{x_i \in \mathbb{R} \mid \exists x_1 \in \mathbb{R}, ..., \exists x_{i-1} \in \mathbb{R}, \exists x_{i+1} \in \mathbb{R}, ..., \exists x_n \in \mathbb{R}, c_j : f_j(\mathbf{x}) = 0\}. \tag{3.40}$$

Elle peut aussi se voir comme la projection $\pi_i(c_j)$ de la contrainte c_j sur le domaine x_i. La figure 3-4 illustre le principe de cette projection pour un ensemble $\mathbb{S}$ donné. Rechercher le plus petit pavé encadrant l'ensemble de solution $\mathbb{S}$ revient donc à déterminer l'ensemble des projections $\pi_i(\mathbb{S})$ sur chacun des domaines associés aux variables x_i. La projection sur x_2 de $\mathbb{S}$ est un ensemble disjoint. Pour encadrer parfaitement cette projection par des intervalles, une union d'intervalles est nécessaire.

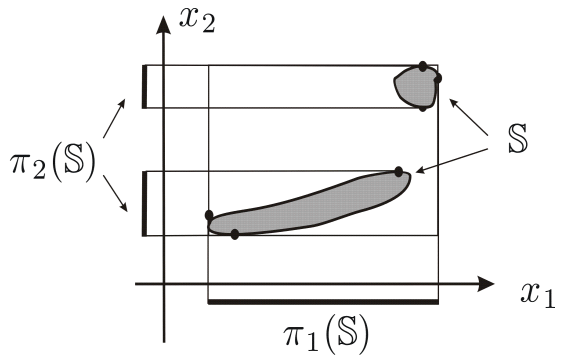

FIG. 3-4 – Projection d'un ensemble $\mathbb{S}$ sur deux axes x_1 et x_2.

Restent alors à construire des opérateurs de projection donnant cette projection. Pour les contraintes dites primitives c'est à dire ne contenant qu'une addition, une multiplication ou une fonction élémentaire, la construction des opérateurs de projection est réalisée en étudiant les variations des fonctions (voir 3.2.4). Ces opérateurs obtiennent, quelques soient les domaines initiaux associés aux variables, la boite optimale enfermant la projection de l'ensemble des points vérifiant la contrainte. Pour les contraintes plus compliquées, une décomposition en contraintes primitives est réalisée (voir 3.2.5).

3.2.2 Notion de consistance

Soit la contrainte

$$x_1 + x_2 = x_3, \tag{3.41}$$

le triplet $(x_1, x_2, x_3) = (1, 2, 4)$ ne satisfait pas la contrainte 3.41, ce point de $\mathbb{R}^3$ est donc *non consistant* avec la contrainte alors que le point $(x_1, x_2, x_3) = (1, 2, 3)$ satisfait la contrainte et est donc *consistant*.

Pour illustrer la consistance sur les domaines continus, revenons sur l'exemple de la figure 3-4. La projection de la contrainte définissant $\mathbb{S}$ nous donne un ensemble sur l'axe x_1. Un intervalle sera dit consistant vis-à-vis de la contrainte c s'il correspond au plus petit intervalle encadrant la projection $\pi_i(c)$. De la même façon sur l'axe x_2, l'intervalle encadrant toutes les parties de l'union, donc encadrant la projection de la contrainte ou de l'ensemble, sera dit consistant.

La figure 3-5 illustre la projection sur les axes x_i des ensembles $\mathbb{S}_1$ et $\mathbb{S}_2$. Prenons les plus petits intervalles contenant les projections $\pi_i(\mathbb{S}_1)$ et $\pi_i(\mathbb{S}_2)$, ils sont représentés par $[x_1]$ et $[x_2]$ sur la figure. Ces deux intervalles sont consistants respectivement pour $\mathbb{S}_1$ et $\mathbb{S}_2$. Prenons maintenant l'intersection de ces deux intervalles représenté par l'intervalle $[x_3]$, elle correspond à un encadrement de la projection $\pi_1(\mathbb{S})$ de l'ensemble $\mathbb{S} = \mathbb{S}_1 \cap \mathbb{S}_2$. Mais cet intervalle n'est pas le plus petit encadrant $\pi_1(\mathbb{S})$, il n'est pas *consistant globalement* mais uniquement *localement*. En effet, certaines valeurs de cet intervalle ne correspondent pas à un point de $\mathbb{S}_1 \cap \mathbb{S}_2$. Il est consistant localement uniquement vis-à-vis des deux contraintes définissant $\mathbb{S}_1$ et $\mathbb{S}_2$. C'est une illustration

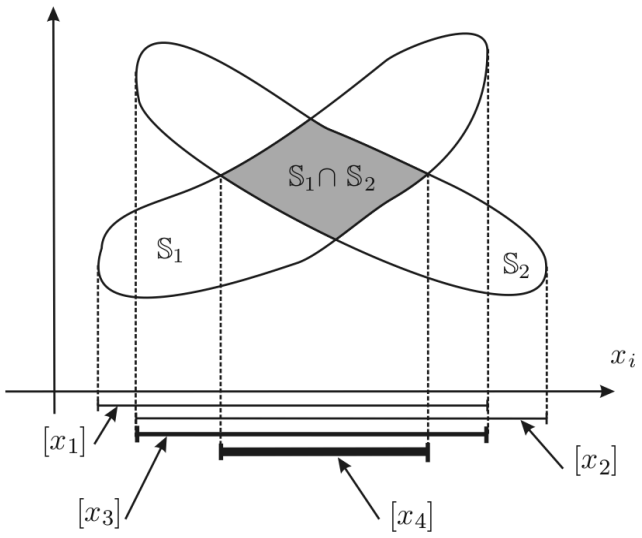

FIG. 3-5 – La projection de deux ensembles $\mathbb{S}_1$ et $\mathbb{S}_2$ permet d'illustrer la consistance globale et la consistance locale.

de la consistance locale. La consistance globale est obtenue avec l'intervalle $[x_4]$ qui contient toutes les valeurs de la projection $\pi_i(\mathbb{S}_1 \cap \mathbb{S}_2)$.

La consistance est aussi liée à la formulation même de la contrainte. Prenons la contrainte suivante

$$c : \frac{(x_1 + x_2).x_2}{x_1} = 0, \tag{3.42}$$

et les domaines

$$x_1 \in [x_1] \text{ et } x_2 \in [x_2]. \tag{3.43}$$

Dans le formalisme CSP nous avons :

$$\mathcal{H} : \left\{ \begin{array}{lcl} \mathcal{V} & = & \{x_1, x_2\}, \\ \mathcal{D} & = & \{[x_1], [x_2]\}, \\ \mathcal{C} & = & \{\frac{(x_1+x_2).x_2}{x_1} = 0\}. \end{array} \right\}. \tag{3.44}$$

Et l'ensemble solution associé :

$$\mathbb{S}_{\mathcal{H}} = \{x_1 \in [x_1], x_2 \in [x_2] \mid \frac{(x_1 + x_2).x_2}{x_1} = 0\}. \tag{3.45}$$

Or cette contrainte fait apparaître des multi-occurrences des variables x_i. Un intervalle $[x_i]$ sera dit *hull consistant* s'il est le plus petit intervalle encadrant la projection de l'ensemble :

$$[x_i] = [\pi_i(\mathbb{S}_{hull})] \tag{3.46}$$

où $\mathbb{S}_{hull}$ est défini par :

$$\mathbb{S}_{hull} = \{a \in [x_1], b \in [x_1], c \in [x_2], d \in [x_2] \mid \frac{(a+c).d}{b} = 0\}, \tag{3.47}$$

Dans cet ensemble chaque occurrence des variables et remplacée par une variable indépendante, la variable x_1 est ainsi remplacée par a et b.

De même, un intervalle $[x_1]$ sera dit *box consistant* s'il est le plus petit intervalle encadrant la projection :

$$[x_1] = [\pi_1(\mathbb{S}_{box1})] \tag{3.48}$$

où $\mathbb{S}_{box}$ est défini par :

$$\mathbb{S}_{box1} = \{x_1 \in [x_1], c \in [x_2], d \in [x_2] \mid \frac{(x_1+c).d}{x_1} = 0\}. \tag{3.49}$$

Pour la hull consistance, chaque occurrence des variables est considérée comme une variable indépendante. Pour la box consistance, le principe est le même sauf pour la variable correspondant à l'axe de projection. Dans notre exemple, x_1 conserve ces multi-occurrences et x_2 reste associée à des variables indépendantes.

La box consistance comme la hull consistance ne permette pas en général d'obtenir la plus petite boîte encadrant l'ensemble $\mathbb{S}_{\mathcal{H}}$. La consistance globale n'est donc pas obligatoirement obtenue.

3.2.3 Notion de contracteur

Les techniques de réduction des domaines intervalles ont des propriétés communes qu'il est intéressant de regrouper dans la notion de contracteur [51]. Le principe est d'augmenter le temps de traitement d'un pavé pour diminuer le temps effectif de résolution des problèmes. Cette diminution du temps effectif de résolution est très liée au problème. Dans le pire des cas l'ajout de ces contracteur diminue l'efficacité de l'algorithme mais dans certains problèmes l'efficacité est augmentée. Appelée aussi filtrage, la notion de contracteur a l'avantage de regrouper un très large éventail de méthodes de réduction des domaines intervalles. Les exemples pour les CSP linéaires sont le contracteur de Gauss Seidel ou le contracteur par programmation linéaire. Pour les CSP non-linéaires, le contracteur de Newton agit sur des systèmes uniquement carrés et différentiables, le contracteur par linéarisation parallèle sur des systèmes rectangulaires et le contracteur par projection de contraintes n'a pas de restriction sur la forme du CSP. Dans la suite de l'étude seul le contracteur utilisant la propagation de contraintes sera détaillé pour suivre ce qui a été codé durant la thèse. Plus de détails sur les autres types de contracteurs peuvent être trouvés dans [51] et [16].

Principe

Le principe du contracteur est d'agir sur un pavé de $\mathbb{IR}^n$ pour éliminer les points non compris dans l'ensemble solution recherché. Le contracteur retourne alors un pavé diminué ou non par

rapport au pavé initial.

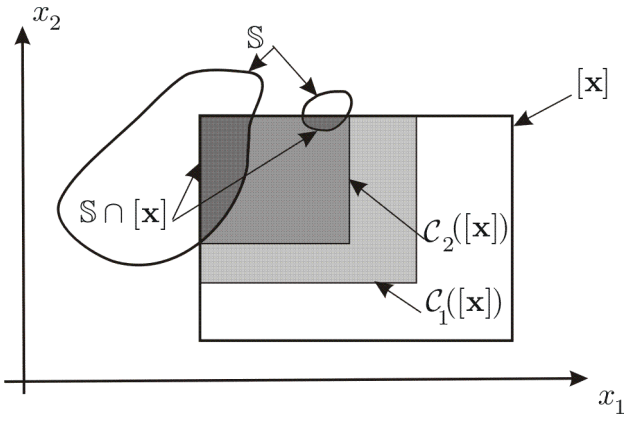

FIG. 3-6 – Illustration d'une contraction optimale sur $[x]$ obtenue avec le contracteur $\mathcal{C}$.

Sur la figure 3-6, l'ensemble des points de $\mathbb{S}$ est l'ensemble des points solutions du problème. Le pavé $[\mathbf{x}]$ est le domaine de recherche. L'ensemble solution correspond donc à l'intersection $\mathbb{S} \cap [\mathbf{x}]$. Quand le contracteur $\mathcal{C}_1$ agit sur $[\mathbf{x}]$, il retourne le pavé $[\mathbf{x}]$ diminué de la zone blanche. L'application d'un deuxième contracteur $\mathcal{C}_2$ à ce nouveau pavé permet d'obtenir le pavé minimal, rectangle qui apparaît en gris foncé sur la figure. Ce pavé contient bien tous les points de $\mathbb{S} \cap [\mathbf{x}]$.

Propriété

Le pavé contracté est toujours inclus dans le pavé de recherche. Cette propriété des contracteurs est appelé contractance :

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, \mathcal{C}([\mathbf{x}]) \subset [\mathbf{x}]. \tag{3.50}$$

L'ensemble des points solutions du pavé de recherche appartient au pavé résultat de la contraction :

$$\forall [\mathbf{x}] \in \mathbb{IR}^n, [\mathbf{x}] \cap \mathbb{S} \subset \mathcal{C}([\mathbf{x}]). \tag{3.51}$$

En général, nous nous intéressons au pavé minimal où toutes les solutions cherchées sont inclues. Le principe de cette recherche est simple tant que le contracteur ne tombe pas sur un pavé d'équilibre, le contracteur $\mathcal{C}$ est une nouvelle fois appliqué au pavé. La contraction s'arrête alors au pavé d'équilibre qui vérifie

$$\mathcal{C}([\mathbf{x}]) = [\mathbf{x}]. \tag{3.52}$$

Le contracteur est donc appelé plusieurs fois. Ces appels successifs peuvent se formaliser par l'opérateur de composition

$$\mathcal{C} \circ \mathcal{C} \circ \mathcal{C}\left([\mathbf{x}]\right) = \mathcal{C}\left(\mathcal{C}\left(\mathcal{C}\left([\mathbf{x}]\right)\right)\right). \tag{3.53}$$

Tant que le pavé $[\mathbf{x}]$ d'équilibre n'est pas obtenu, appliquer une nouvelle fois le contracteur $\mathcal{C}$ permet de réduire les domaines intervalles. Le nombre d'appels du contracteur est donc très lié au problème et difficile à déterminer à priori.

Exemple

L'exemple de la figure 3-7 présente deux contracteurs $\mathcal{C}_{\mathbb{S}_1}$ et $\mathcal{C}_{\mathbb{S}_2}$ réalisant la projection des contraintes associées à $\mathbb{S}_1$ et $\mathbb{S}_2$ sur les deux axes. de chacun des ensembles de solutions. Les

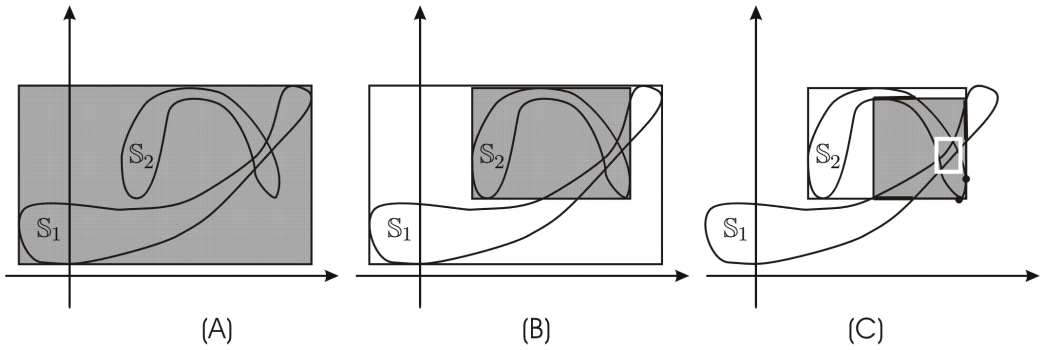

FIG. 3-7 – Illustration de la composition de deux contracteurs $\mathcal{C}_{\mathbb{S}_1}$ et $\mathcal{C}_{\mathbb{S}_2}$ avec représentation du pavé optimal en blanc.

contracteurs sont appliqués suivant la séquence :

$$\mathcal{C}_{\mathbb{S}_1} \circ \mathcal{C}_{\mathbb{S}_2} \circ \mathcal{C}_{\mathbb{S}_1}([\mathbf{x}]) = \mathcal{C}_{\mathbb{S}_1}(\mathcal{C}_{\mathbb{S}_2}(\mathcal{C}_{\mathbb{S}_1}([\mathbf{x}]). \tag{3.54}$$

Les trois étapes de cette séquence sont visibles sur la figure 3-7 où chaque pavé gris représente le pavé obtenu après contraction. Sur la figure 3-7C, le pavé gris est un point fixe pour les deux contracteurs. En effet sur chacune des arrêtes du pavé fixe, il existe un point appartenant aux ensembles solutions $\mathbb{S}_1$ et $\mathbb{S}_2$. Mais ce pavé n'est pas la boîte minimale, celle-ci correspond au rectangle blanc sur la figure.

3.2.4 Contraintes primitives ou opérateur de projection

Les algorithmes de propagation reposent sur la projection optimale des contraintes dites primitives. Ces contraintes de 1, 2 ou 3 variables (unaire, binaire ou ternaire) correspondent à des contraintes contenant des opérateurs arithmétiques, des fonctions (trigonométrique, exponentielle,...) qui relient les variables entre elles. Les opérateurs de projection ainsi que la façon de les implémenter sont détaillés dans cette partie. Nous nommons ces opérateurs de projection grâce à un C majuscule pour signifier que c'est une contrainte. L'opération ou la fonction associées à la contrainte primitive est représentée par l'exposant (+,-,*,/,sin,cos,...).

Contrainte primitive des opérateurs arithmétiques

L'algorithme de projection de l'addition est donné par l'algorithme C^+. La contrainte associée est

$$x_3 = x_1 + x_2. \tag{3.55}$$

Algorithme C^+ (entrée : *sens* entrées sorties : $[x_3], [x_1], [x_2]$)	
1	si $sens = 1$
2	$[x_3] := [x_3] \cap ([x_1] + [x_2])$;
3	si $sens = -1$
4	$[x_1] := [x_1] \cap ([x_3] - [x_2])$;
5	$[x_2] := [x_2] \cap ([x_3] - [x_1])$;

Dans l'algorithme deux sens sont possibles, 1 et -1, correspondant à la propagation dans le sens 1 et à la rétro-propagation dans le sens -1. La projection de la contrainte sur l'ensemble des domaines des variables est évitée lorsqu'elle n'est pas nécessaire. Par exemple pour la propagation seule la contraction du domaine $[x_3]$ nous interresse parce que l'information transite de x_1, x_2 vers x_3 d'où le sens 1.

Pour la contrainte soustraction, l'opérateur de projection est le suivant :

Algorithme C^- (entrée : *sens* entrées sorties : $[x_3], [x_1], [x_2]$)	
1	si $sens = 1$
2	$[x_3] := [x_3] \cap ([x_1] - [x_2])$;
3	si $sens = -1$
4	$[x_1] := [x_1] \cap ([x_3] + [x_2])$;
5	$[x_2] := [x_2] \cap ([x_1] - [x_3])$;

Exemple 3.2.1 *Soit l'ensemble*

$$\mathbb{S} = \{(x_1, x_2, x_3) \in \mathbb{R}^3 |\ x_3 = x_2 - x_1\}. \tag{3.56}$$

En prenant

$$[\mathbf{a}] = [1; 2] \times [1; 6] \times [1; 3], \tag{3.57}$$

$$[\mathbf{b}] = [4; 7] \times [2; 6] \times [1; 3], \tag{3.58}$$

nous obtenons

$$\mathcal{C}^*_{\mathbb{S}}([\mathbf{a}]) = [1; 2] \times [2; 5] \times [1; 3], \tag{3.59}$$

$$\mathcal{C}^*_{\mathbb{S}}([\mathbf{b}]) = [4; 5] \times [5; 6] \times [1; 2], \tag{3.60}$$

Le pavé obtenu après contraction de $[\mathbf{a}]$ *et de* $[\mathbf{b}]$ *par* $\mathcal{C}^*_{\mathbb{S}}$ *est le plus petit pavé contenant toutes les solutions appartenant à* $\mathbb{S}$. *La figure 3-8 illustre la contraction des pavés* $[\mathbf{a}]$ *et* $[\mathbf{b}]$.

Pour les opérateurs de projection de la multiplication et la division, une version simpliste peut être exprimée de la façon suivante :

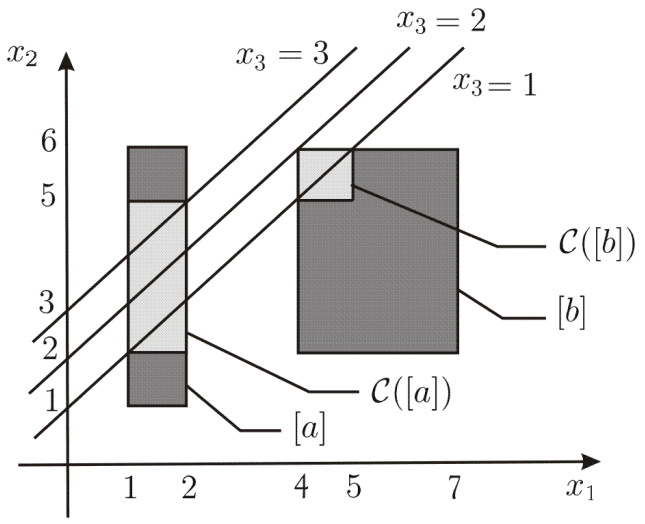

FIG. *3-8 – Contraction de la contrainte $x_3 = x_2 - x_1$ sur les pavés* [**a**] *et* [**b**].

Algorithme C*(entrée : *sens* entrées sorties : $[x], [y], [z]$)	
1	si $sens = 1$
2	$[z] := [z] \cap ([x] * [y])$;
3	si $sens = -1$
4	$[x] := [x] \cap ([z]/[y])$;
5	$[y] := [y] \cap ([z]/[x])$;

Prenons l'exemple suivant :

$$[x] \times [y] \times [z] = [0; 5] \times [-5; 5] \times [1; 2],$$

l'algorithme de projection C* présenté ci-dessus ne permet pas de réduire les domaines alors que la plus petite boîte encadrant l'ensemble de solutions (le plus petit pavé consistant) est $[\frac{1}{5}; 5] \times [\frac{1}{5}; 5] \times [1; 2]$. Le problème vient du fait que 0 appartient aux domaines $[x]$ et $[y]$, et intervient dans les divisions des étapes 4 et 5.

Pour obtenir la projection optimale, l'algorithme à mettre en oeuvre est plus élaboré. Une proposition d'algorithme est présentée avec les trois algorithmes C^{*optimal}, C$^{*sens-1}$ et C$^{*cadrant++}$. La démarche suivie est la suivante : i) découpage du domaine de variation en partie simple (si possible monotone ou de résolution immédiate) ii) projection de la contrainte sur ces parties iii) réunion des différentes projections. Sur la figure 3-9 les trois schémas représentent le découpage que nous avons effectué. La contraction n'est pas optimale dans les deux cas $z^- > 0$ et $z^+ < 0$ le troisième cas étant traité de façon classique au étapes 2 et 3 de l'algorithme C$^{*sens-1}$. Pour les deux cas qui posent problème, l'étude se ramène à la projection dans deux quadrants de l'espace $\mathbb{R}^2$: dans le cas $z^- > 0$ les quadrants $[0; +\infty[\times[0; +\infty[$ et $]-\infty; 0]\times]-\infty; 0]$, dans le cas $z^+ < 0$ les quadrants $]-\infty; 0] \times [0; +\infty[$ et $[0; +\infty[\times]-\infty; 0]$. Nous pouvons voir sur la figure 3-9, que dans les deux premiers cas, seuls deux quadrants comportent des points solutions. Une

illustration de contraction optimale est présentée pour chacun des cas (le pavé initial étant en gris foncé et le pavé obtenu en gris clair).

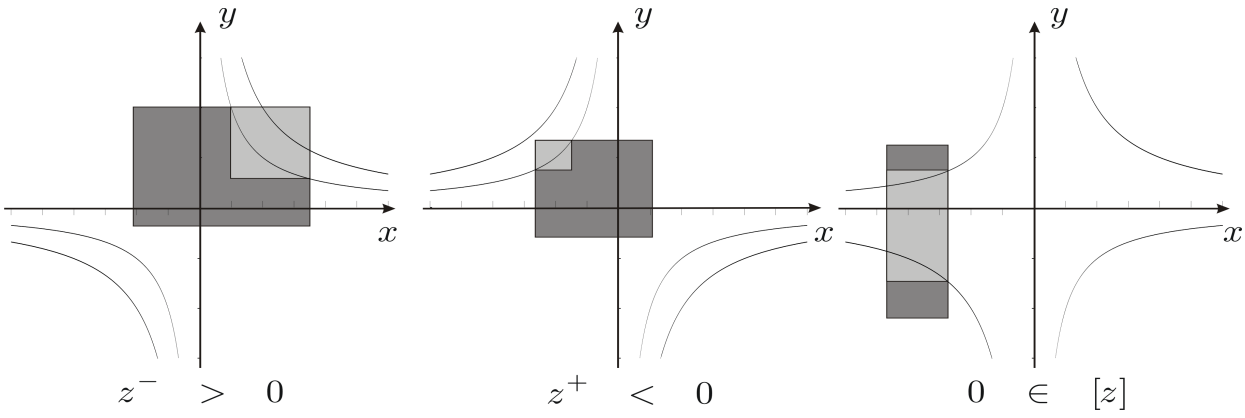

FIG. 3-9 – Présentation de la séparation en trois cas de la contrainte primitive $z = x.y$ pour le codage de l'opérateur de projection.

Algorithme C$*^{optimal}$ (entrée : *sens* entrées sorties : $[x], [y], [z]$)	
1	si $sens = 1$
2	$[z] := [z] \cap ([x] * [y])$;
3	si $sens = -1$
4	C$*^{sens-1}$ $([x], [y], [z])$;

L'algorithme C* est donc modifié et remplacé par C$*^{optimal}$. Le traitement de la projection sur les variables x et y correspondant à la rétro-propagation est effectué par l'algorithme C$*^{sens-1}$.

Algorithme C$*^{sens-1}$ (entrées sorties : $[x], [y], [z]$)	
1	si $(0 \in [z])$
2	$[x] := [x] \cap ([z]/[y])$;
3	$[y] := [y] \cap ([z]/[x])$;
4	sinon
5	$[x_d] := [x] \cap [0; +\infty[$; $[y_d] := [y_d] \cap [0; +\infty[$;
6	$[x_g] := [x] \cap]-\infty; 0]$; $[y_g] := [y_g] \cap]-\infty; 0]$;
7	si $(z^- > 0)$
8	C$*^{cadrant++}$ $([x_d], [y_d], [z])$;
9	C$*^{cadrant++}$ $(-[x_g], -[y_g], [z])$;
10	si $(z^+ < 0)$
11	C$*^{cadrant++}$ $(-[x_g], [y_d], -[z])$;
12	C$*^{cadrant++}$ $([x_d], -[y_g], -[z])$;

Enfin l'algorithme C$*^{cadrant++}$ réalise la projection sur les variables x et y pour $x \in \mathbb{R}^+$ et $y \in \mathbb{R}^+$.

Algorithme C$^{*cadrant++}$ (entrées sorties : $[x_d], [y_d], [z]$)	
1	si $(x_d^+ < 0)\&(y_d^+ < 0)$
2	si $(y_d^+ \neq +\infty)$ $[x_d] := [x_d] \cap [\frac{z^-}{y_d^+}; +\infty[$;
3	si $(x_d^+ \neq +\infty)$ $[y_d] := [y_d] \cap [\frac{z^-}{x_d^+}; +\infty[$;
4	sinon $[x_d] := \varnothing$; $[y_d] := \varnothing$;
5	si $(x_d^- < 0)$ $[y_d] := [y_d] \cap [0; \frac{z^+}{x_d^-}]$;
6	si $(y_d^- < 0)$ $[y_d] := [y_d] \cap [0; \frac{z^-}{y_d^-}]$;

Remarque 3.2.1 *L'algorithme de calcul de la division optimale est plus long du fait des intersections et des unions des parties gauches et droites pour $[x]$ et $[y]$. Dans certains problèmes où la présence de 0 dans les domaines de la multiplication n'intervient pas, il peut être souhaitable pour la rapidité des calculs de conserver l'implémentation simple de la division. Le choix revient à l'utilisateur et dépend de sa connaissance du problème.*

Contrainte primitive $y = f(x)$ où f est une fonction continue monotone

Soit la fonction $f : \mathbb{D} \longrightarrow \mathbb{D}$, la contrainte associée à cette fonction est $y = f(x)$, x et y prenant pour domaines $x \in [x^-; x^+]$ $y \in [y^-; y^+]$. Si f est monotone et continue sur $\mathbb{D}$, il existe $a \in \mathbb{D}$ et $b \in \mathbb{D}$ tel que $f(a) = y^-$ et $f(b) = y^+$. Les plus petits domaines pour x et y s'obtiennent de la façon suivante :

$$\begin{array}{ll} [x] := [x] \cap [a, b] \text{ si } f \text{ est croissante} & [x] := [x] \cap [b, a] \text{ sinon} \\ [y] := [y] \cap [f(x^-); f(x^+)] & [y] := [y] \cap [f(x^+); f(x^-)] \end{array} \tag{3.61}$$

Nous avons codé plusieurs fonctions de ce type : l'exponentielle $y = e^x$, le logarithme $y = \ln(x)$, les puissances entières impaires et positives $y = x^{2n+1}$ $n \in \mathbb{N}$, la fonction racine $y = \sqrt{x}$.

Contrainte primitive $y = f(x)$ où f est une fonction continue monotone

Quand la fonction change de monotonie une seule fois sur son domaine de définition, le traitement de l'opérateur de projection s'effectue alors sur les domaines où la fonction est monotone.

Sur l'exemple de la fonction valeur absolue le traitement est séparé sur les parties $\mathbb{R}^-$ et $\mathbb{R}^+$.

Algorithme C$^{\text{ABS}}$ (entrée : *sens* entrées sorties : $[y], [x]$)	
1	$[x]_g := [x] \cap \mathbb{R}^-$; $[x]_d := [x] \cap \mathbb{R}^+$;
2	si $sens = 1$ $[y]_g := [y] \cap (-[x]_g)$; $[y]_d := [y] \cap [x]_d$;
3	$[y] := [y]_g \cup [y]_d$;
4	si $sens = -1$ $[x]_g := [x]_g \cap [y]$; $[x]_d := [x]_d \cap (-[y])$;
5	$[x] := [x]_g \cup [x]_d$;

Les fonctions de ce type que nous avons codées sont les suivantes : les puissances entières paires et positives $y = x^{2n}$ $n \in \mathbb{N}$, la fonction valeur absolue $y = |x|$.

Quand la fonction change plusieurs fois de monotonie, le traitement est alors plus difficile. Parmi ces fonctions, les fonctions périodiques, notamment les fonctions trigonométriques permettent de définir un opérateur de projection dont le principe est le suivant. A l'aide de symétries et de translations, chaque partie du domaine d'étude est ramenée à un domaine où la fonction est monotone. Les périodes se trouvant aux extrémités droite et gauche de $[x]$ sont ainsi projetées et nous donnent grâce à l'union de ces deux boites, la boite minimale.

Exemple 3.2.2 *Considérons maintenant l'ensemble*

$$\mathbb{S} = \{(x, y) \in \mathbb{R}^2 | y = \sin(x)\}. \tag{3.62}$$

Les pavés $[\mathbf{a}], \ldots, [\mathbf{f}]$ *et* $[\mathbf{e}]$ *sont représentés en gris foncé sur la figure 3-10, les pavés contractés* $\mathcal{C}^*_{\mathbb{S}}([x] \times [y])$ *sont en gris clair. Appliquer le contracteur* $\mathcal{C}^*_{\mathbb{S}}$ *sur un pavé* $[x] \times [y]$ *génère le plus petit pavé contenant l'ensemble des points appartenant à* $\mathbb{S} \cap [x] \times [y]$*. Sur la figure 3-10, différentes configurations sont représentées (i) le pavé* $[\mathbf{e}]$ *n'intersecte aucun point de la contrainte sinus et donc* $\mathcal{C}^*_{\mathbb{S}}([\mathbf{e}]) = \emptyset$*, (ii) le pavé* $[\mathbf{f}]$ *ne peut être contracté* $\mathcal{C}^*_{\mathbb{S}}([\mathbf{f}]) = [\mathbf{f}]$*, (iii) les pavés* $[\mathbf{b}]$ *et* $[\mathbf{c}]$ *ne peuvent être contractés que suivant un axe et (iv) les pavés* $[\mathbf{a}]$ *et* $[\mathbf{d}]$ *qui peuvent être contractés suivant les deux directions.*

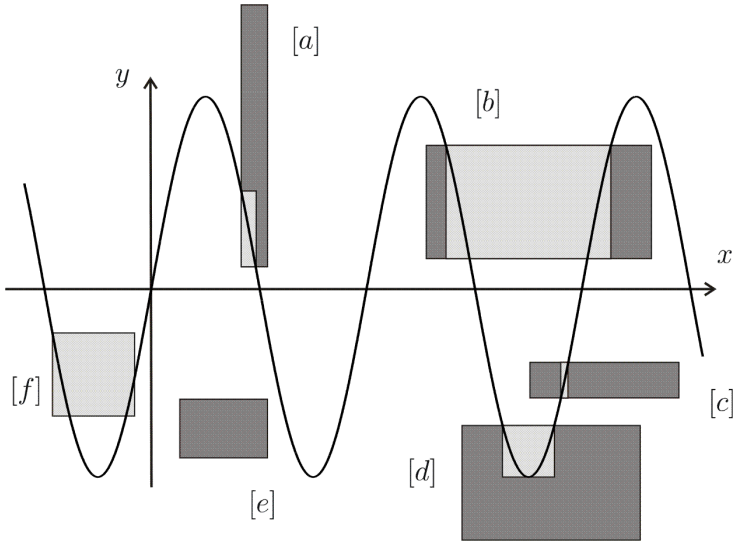

FIG. *3-10 – Opérateur de contraction de la contrainte sinus.*

Les fonctions de ce type que nous avons codées : les fonctions trigonométriques $y = \sin(x)$, $y = \cos(x)$ et $y = \tan(x)$.

Contrainte primitive $y = f(x)$ où f est une fonction non continue

Certaines fonctions utilisées dans les problèmes d'automatique et permettant le calcul de dérivées généralisées sont des fonctions nécessitant le codage d'un opérateur de projection. La fonction

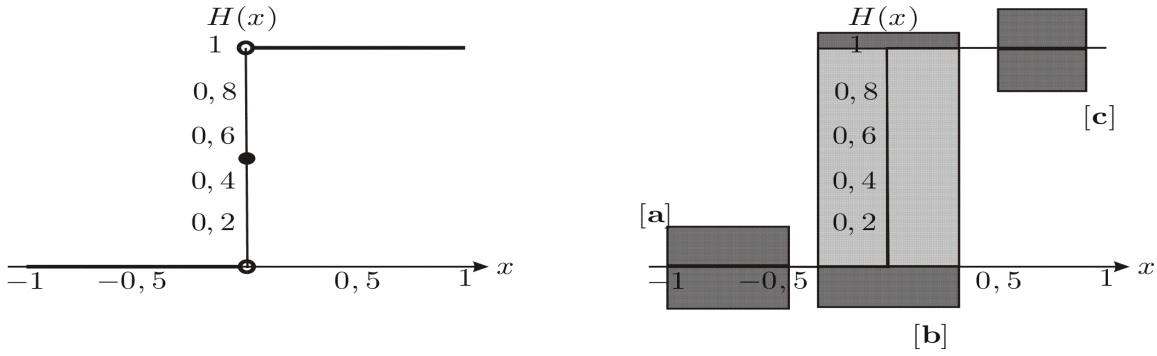

FIG. 3-11 – Schéma représentant la fonction heaviside et l'opérateur de projection pour la contrainte $y = heaviside(x)$ associée.

Heavyside encore appelée step est définie par :

$$H(x) = \begin{cases} 0 \text{ si } x < 0 \\ \frac{1}{2} \text{ si } x = 0 \\ 1 \text{ si } x > 0 \end{cases} . \tag{3.63}$$

L'opérateur de projection associé est représenté sur la figure 3-11(B). La projection de la contrainte visible sur y renvoie alors $[0]$, $[0;1]$ et $[1]$ respectivement pour les domaines $[x_1] < 0$, $[x_2]$ contenant 0 et $[x_3] > 0$. La projection sur x contractant les domaines à la partie négative quand $[y] = [0]$ et à la partie positive quand $[y] = [1]$.

D'autres contraintes sont aussi envisageables. Le codage de l'opérateur de projection de la fonction pic de Dirac peut s'effectuer de la manière suivante

$$\begin{array}{ll} [y] := [y] \cap [0; +\infty[& \text{si } 0 \in [x], \\ [y] := [y] \cap [0] & \text{si } 0 \notin [x], \\ [x] := [x] \cap [0] & \text{si } 0 \notin [y]. \end{array} \tag{3.64}$$

Enfin la connaissance de certaines propriétés sur les variables peut être un facteur de contraction supplémentaire, par exemple dans les problèmes où certaines variables sont entières ou booléennes.

Exemple 3.2.3 *Soit la variable $n \in \mathbb{Z}$ une variable entière, si le domaine associé à cette variable est $[0,5; 5,26]$ la contrainte entiere(x) permet de réduire le domaine à $[1;5]$. De la même façon pour une variable booléenne $b \in \{0,1\}$ qui varie dans les domaines $[0;0,9]$, $[0,1;1]$ et $[0,1;0,9]$ permet de réduire ceux-ci respectivement $[0]$, $[1]$ et $\varnothing$.*

Remarque 3.2.2 *Bien que ces deux opérateurs de contraction permettent de réduire les domaines des variables entières ou booléennes, la résolution de problèmes impliquant variables entières et continues nécessite des algorithmes spécifiques.*

Contrainte primitive $y = f(\mathbf{x})$ où f est une fonction multi-variables

Les contraintes du type :

$$y = \min(x_1, x_2, ..., x_n) \tag{3.65}$$

$$y = \max(x_1, x_2, ..., x_n) \tag{3.66}$$

se traitent de la même façon. L'opérateur de projection de la contrainte :

$$c = \max(a, b), \tag{3.67}$$

étant utilisé comme une fonction composée

$$y = \max(x_1, \max(x_2, \max(..., \max(x_{n-1}, x_n)))). \tag{3.68}$$

Sur la figure 3-12, la contrainte $y = \max(x_1, x_2)$ est tracée sur l'espace (x_1, x_2) quand y vérifie

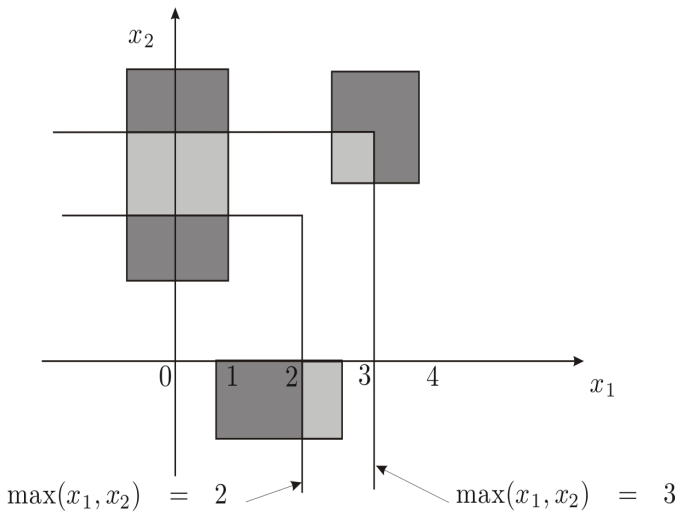

FIG. 3-12 – Opérateur de projection de la contrainte max, les pavés initiaux sont en gris foncé, les pavés contractés en gris clair.

$y \in [2; 3]$. La contraction des trois pavés gris foncés $[\mathbf{x}]$ vérifiant la contrainte est visible, les pavés obtenus sont en gris clair. L'opérateur de contraction pour cette contrainte peut être donné par l'algorithme C^{MAX}

Algorithme $\mathrm{C}^{\mathrm{MAX}}$ (entrée : *sens* entrées sorties : $[z], [x], [y]$)	
1	si $sens = 1$ $\quad [z] := [z] \cap (\max([x], [y]);$
2	si $sens = -1$
3	si $((x^+ < z^-)$ ou $(z^+ < x^-))$ ($[z]$ et $[x]$ disjoint)
4	$[y] := [y] \cap [z];$
5	si $((y^+ < z^-)$ ou $(z^+ < y^-))$ ($[z]$ et $[y]$ disjoint)
6	$[x] := [x] \cap [z];$
	$[x] := [x] \cap]-\infty; z^+];$
7	$[y] := [y] \cap]-\infty; z^+];$

3.2.5 Algorithme de propagation de contraintes

Les opérateurs de projection pour les contraintes primitives permettent de mettre en place les algorithmes de propagation de contraintes. De nombreux algorithmes de propagation de contraintes existent [11]. Mais pour toutes ces méthodes, le principe suit les points suivants.

Projection. Pour un ensemble $\mathbb{S}$ fermé, notons $\mathcal{C}^*_{\mathbb{S}}([\mathbf{x}])$ le plus petit pavé (c'est-à-dire le produit cartésien d'intervalles réels) contenant $\mathbb{S} \cap [\mathbf{x}]$. $\mathcal{C}^*_{\mathbb{S}}$ est appelé *contracteur* (car il enferme $\mathbb{S} \cap [\mathbf{x}]$) *optimal* (il n'en existe pas de plus petit, repéré par l'étoile en exposant) associé à l'ensemble $\mathbb{S}$. Le calcul de $\mathcal{C}^*_{\mathbb{S}}$ est communément appelé *projection.* Un ensemble pour lequel nous disposons d'un contracteur optimal est appelé *ensemble primitif.* Les exemples de projection pour des ensembles $\mathbb{S}$ primitifs sont ceux vus précédemment pour les fonctions $\sin(x)$, x^n, $x_1 \Diamond x_2$ où $\Diamond \in \{+, -, *, /, \min, \max\}$, ...

Propagation. Afin d'illustrer la propagation de contraintes, considérons maintenant l'ensemble $\mathbb{S}$ défini par la conjonction de deux contraintes c_1 et c_2.

$$\mathbb{S} \triangleq \left\{(x_1, x_2) \in \mathbb{R}^2 \mid c_1 : x_1 = \sin(x_2) \text{ et } c_2 : x_1 = x_2^3,\right\}$$

Cherchons des intervalles, si possible petits, qui contiennent toutes les valeurs possibles pour x_1 et x_2. L'ensemble $\mathbb{S}$ peut-être défini comme l'intersection des deux ensembles primitifs :

$$\mathbb{S}_1 = \{(x_1, x_2) | x_1 = \sin(x_2)\}, \tag{3.69}$$

$$\mathbb{S}_2 = \{(x_1, x_2) | x_1 = (x_2)^3\}. \tag{3.70}$$

pour lesquels nous disposons de contracteurs $\mathcal{C}^*_{\mathbb{S}_1}$ et $\mathcal{C}^*_{\mathbb{S}_2}$ optimaux. Nous obtenons la contraction suivante :

$$\mathcal{C}^*_{\mathbb{S}_1}([\mathbf{x}]) = [-1; 1] \times]-\infty; +\infty[\tag{3.71}$$

puis :

$$\mathcal{C}^*_{\mathbb{S}_2}\left(\mathcal{C}^*_{\mathbb{S}_1}([\mathbf{x}])\right) = [-1; 1] \times [-1; 1] \tag{3.72}$$

et encore :

$$\mathcal{C}^*_{\mathbb{S}_2}\left(\mathcal{C}^*_{\mathbb{S}_2}\left(\mathcal{C}^*_{\mathbb{S}_1}([\mathbf{x}])\right)\right) = [-0,84; 0,85] \times [-1; 1]. \tag{3.73}$$

Nous assistons donc pas à pas à la contraction du pavé de $[\mathbf{x}]$. Les pavés obtenus à la suite

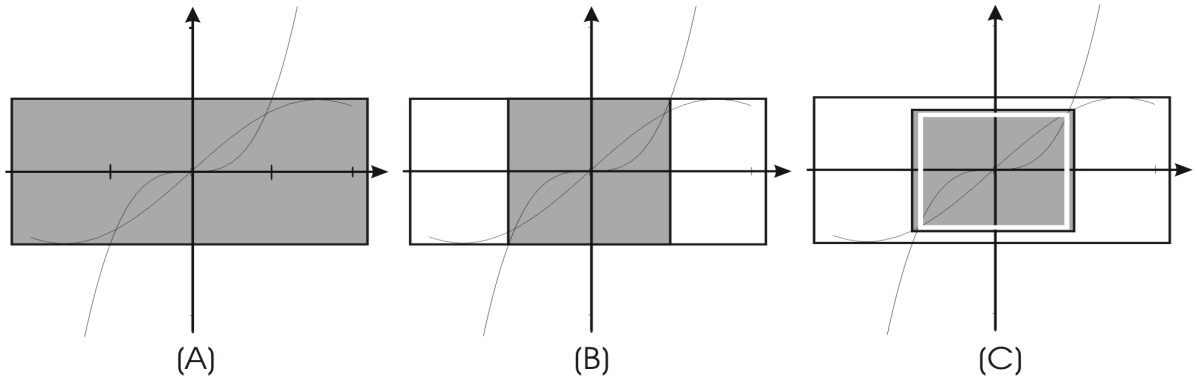

FIG. 3-13 – Représentation des trois premières étapes de propagation de contraintes

de trois contractions sont visualisés en gris sur le schéma 3-13. Le principe de propagation de contraintes est donc de composer itérativement les contracteurs $\mathcal{C}^*_{\mathbb{S}_2}$ et $\mathcal{C}^*_{\mathbb{S}_1}$ afin d'obtenir un contracteur $\mathcal{C}_{\mathbb{S}}$ pour $\mathbb{S} = \mathbb{S}_1 \cap \mathbb{S}_2$. La composition des contracteurs optimaux $\mathcal{C}^*_{\mathbb{S}_1}$ et $\mathcal{C}^*_{\mathbb{S}_2}$ s'arrête à l'obtention d'un point fixe $[\bar{\mathbf{x}}]$ qui satisfait :

$$\mathcal{C}^*_{\mathbb{S}_1}([\bar{\mathbf{x}}]) = \mathcal{C}^*_{\mathbb{S}_2}([\bar{\mathbf{x}}]) = [\bar{\mathbf{x}}]. \tag{3.74}$$

Attention, même si $\mathcal{C}^*_{\mathbb{S}_1}$ et $\mathcal{C}^*_{\mathbb{S}_2}$ sont tous les deux optimaux, le contracteur $\mathcal{C}_{\mathbb{S}}$ n'est pas nécessairement optimal. En effet, les plus petits intervalles contenant toutes les solutions sont représentés par le rectangle blanc de la figure 3-13(C) et la propagation de contraintes s'arrête au point fixe $[\bar{\mathbf{x}}]$ parfois avant l'obtention de cette boite minimale.

Décomposition des contraintes. Les contraintes de notre problème d'étalonnage sont généralement complexes et il faut effectuer une décomposition. Afin d'illustrer cette décomposition, considérons les contraintes :

$$y_1 = \cos(i_1 + i_2).\sin(i_1 + i_2), \tag{3.75}$$
$$y_2 = i_3.\sin^2(i_1 + i_2). \tag{3.76}$$

Ces dernières peuvent être décomposées en contraintes primitives de la façon suivante :

$$\begin{array}{ll} a_1 = i_1 + i_2, & \\ a_2 = \cos(a_1), & a_5 = i_1 + i_2, \\ a_3 = i_1 + i_2, & a_6 = \sin(a_5), \\ a_4 = \sin(a_3), & a_7 = a_6^2, \\ y_1 = a_2.a_4. & y_2 = i_3.a_7. \end{array} \tag{3.77}$$

Ne disposant d'aucune connaissance sur les variables intermédaires $a_{1\ldots4}$, nous leur associons le domaine $]-\infty;+\infty[$.

Remarque 3.2.3 *Lors de l'implémentation, ces domaines doivent être stockés en mémoire pour conserver à chaque étape l'information sur les domaines. Ce stockage est lié à la structure des contraintes dans le programme : liste de contraintes primitives, arbres représentant les contraintes...*

Remarque 3.2.4 *Quelque soit l'ordre de projection des contraintes primitives le point fixe obtenu reste le même. Seul le temps de résolution est différent. Pour une fonction $y = f(\mathbf{x})$ où les variables x_i ne sont occurrentes qu'une seule fois, le point fixe optimal est obtenu. Pour l'obtenir avec le temps le plus court, le principe de propagation rétro-propagation est le plus performant. La propagation commence la résolution avec le traitement des contraintes primitives contenant les variables $\mathbf{x}$, puis des variables intermédiaires associées pour finir par la contrainte primitive liant l'expression à y. L'ordre de résolution inverse est appliqué lors de la rétro-propagation. Une seule propagation rétro-propagation est nécessaire dans le cas de contrainte unique avec variables non multi-occurrentes. Dans le cas général, avant d'atteindre le pavé d'équilibre, plusieurs propagations rétro-propagations sont nécessaires.*

Remarque 3.2.5 *La décomposition en contraintes primitives n'est pas unique mais permet l'obtention d'un même pavé d'équilibre.*

En résumé. Pour effectuer une propagation de contraintes, nous décomposons les contraintes en contraintes primitives. Enfin, nous projetons chacune des contraintes primitives jusqu'au pavé d'équilibre.

3.2.6 Quelques problèmes

Bilan

La recherche du point fixe optimal ou de la plus petite boîte encadrant l'ensemble des solutions est bien sûr l'objectif mais souvent des problèmes de consistance locale empêchent son obtention.

Dans l'exemple 3.1.3, les fonctions d'inclusion ne permettent pas l'obtention de la fonction d'inclusion minimale, la forme $f_1(x) = 2.x.x - 2.x - 1$ qui a le plus d'occurrences de x étant la moins performante, la forme de Horner $f_3(x) = x.(2.x - 2) - 1$ étant la plus performante. Si maintenant la contrainte :

$$y = f(x) \tag{3.78}$$

doit être vérifiée avec $y \in [0; 0.5]$ et $x \in [0; 3]$ alors la propagation de contraintes va permettre de contracter les domaines mais son comportement sera lié à la formulation de $f(x)$. La formulation

$f_2(x) = 2.x^2 - 2.x - 1$ apportant la contraction la plus forte :

$$\begin{array}{ll} \text{formulation} & \text{domaine obtenu} \\ f_1 & x \in [0,249; 3] \\ f_3 & x \in [1,318; 1,569] \\ f_2 & x \in [1,365; 1,51] \end{array} . \tag{3.79}$$

Une amélioration de la contraction peut être obtenue en utilisant le calcul formel, comme par exemple l'utilisation des bases de grobner étudiées dans [29].

Un autre problème, comme nous l'avons vu pour la contrainte $z = x.y$, est le fait que les opérateurs de projection des ensembles primitifs doivent veiller à être optimaux. En effet le fait de ne pas avoir la boîte minimale peut nécessiter des propagations rétro-propagations supplémentaires rallongeant le temps de calcul ou engendrer un pavé fixe que les algorithmes de propagation ne peuvent lever. La contrainte primitive $y = \text{sinc}(x)$ peut être vue comme une contrainte non primitive

$$y = \frac{\sin(x)}{x}, \tag{3.80}$$

mais dans ce cas, l'obtention de la projection optimale est difficile avec la présence du point singulier en 0. La projection nécessite alors le recours à une étude de la fonction.

Complexité

La complexité des algorithmes est le nombre d'opérations effectuées dans le cas le plus défavorable. Ici l'algorithme contracte le pavé de dimension n et de taille $\omega([\mathbf{x}])$. Si la contraction est inférieure à la précision ε, l'algorithme s'arrête.

Le cas le plus défavorable est donc quand la contraction ne s'effectue que sur une dimension de $[\mathbf{x}]$ et avec une profondeur de ε. Il y a alors $n\frac{\omega([\mathbf{x}])}{\varepsilon}$ propagations rétro-propagations..

La contraction s'effectue quand une des contraintes élémentaires réussit à réduire le pavé $[\mathbf{x}]$. Dans le cas le plus défavorable, une seule contrainte élémentaire sur les m contraintes effectue une contraction. Le nombre d'appels au programme de contractions élémentaires est donc m fois le nombre de propagations rétro-propagations..

La complexité est donc de la forme suivante :

$$Complexité = n.m.\frac{\omega([\mathbf{x}])}{\varepsilon} \tag{3.81}$$

C'est un résultat très important car une complexité pseudo-polynomiale est caractéristique d'algorithme performant mais pouvant être soumis à des phénomènes de convergence lente.

Exemple 3.2.4 *Prenons les deux contraintes :*

$$y = x \tag{3.82}$$
$$y = -2.x \tag{3.83}$$

Si le domaine pour x est $[-10;10]$ le domaine pour y est $]-\infty;+\infty[$, les étapes de propagations rétro-propagations vont être les suivantes :

$$\begin{aligned} 1p &: [y] :=]-\infty;+\infty[\cap[-10;10] = [-10;10] \\ 1rp &: [x] := [-10;10] \cap [-5;5] = [-5;5] \\ 2p &: [y] := [-10;10] \cap [-5;5] = [-5;5] \\ 2rp &: [x] := [-5;5] \cap [-2.5;2,5] = [-2.5;2.5] \\ &... \end{aligned}$$

Au bout d'une vingtaine de propagations rétro-propagations, l'intervalle atteint une taille inférieure à un ϵ donné, mais ne permet jamais d'obtenir les intervalles $[0]\times[0]$ véritable solution du CSP. Remplaçons maintenant le coefficient 2 par 1.0001, le nombre d'étapes explose, la méthode est dans la configuration d'une convergence lente (algorithme de complexité quasi-polynomial). C'est sans doute la proximité du problème avec le CSP défini par les contraintes :

$$y = x,\ y = -x \tag{3.84}$$

avec $]-\infty;+\infty[\times]-\infty;+\infty[$ pour domaines qui engendre ce phénomène. En effet ce CSP a une infinité de points fixes de la forme $[-a;a]\times[-a;a]$.

3.2.7 Contraintes globales

Une des premières améliorations apportée aux algorithmes de projection est l'élargissement de la classe d'opérateurs de projection optimaux. Les contraintes associées ne sont plus appelées primitives mais correspondent plutôt à la notion de contrainte globale. Ce sont des contraintes que la propagation de contraintes (utilisant les opérateurs de projection de contraintes primitives) n'arrive pas à contracter à la boîte minimale. Néanmoins d'autres techniques peuvent permettre d'obtenir rapidement cette boîte.

Principe avec la contrainte parabole

Prenons la contrainte :

$$y = a.x^2 + b.x + c, \tag{3.85}$$

si $(a,b,c) \in \mathbb{R}^3$, la connaissance des domaines pour x et y permet de créer un opérateur de projection spécifique. La dérivé s'annulant en $x_m = -\frac{b}{2.a}$ nous obtenons le point d'inflexion avec $y_m = c - \frac{b^2}{4a}$, connaissant le signe de a, la monotonie de la fonction est déterminée. Nous avons

donc un opérateur qui permet d'obtenir les plus petits intervalles pour x et pour y de façon garantie.

Remarque 3.2.6 *Il est aussi possible de construire cet opérateur de projection avec des coefficients intervalles* $(a, b, c) \in \mathbb{IR}^3$.

Remarque 3.2.7 *Il est possible d'étendre la classe des opérateurs de projection à l'ensemble des polynômes de degré* 6*, les racines d'un polynôme de degré* 5 *étant disponibles analytiquement.*

Remarque 3.2.8 *Pour que ces contraintes globales polynomiales soient utilisables, la nécessité d'une reconnaissance automatique de polynômes parmi les contraintes doit être effectuée. Les problèmes étant généralement multi-variables, des manipulations formelles sont à envisager.*

Contrainte angle

La rotation d'un angle θ selon un axe peut être décrite à l'aide d'une matrice de rotation. Cette matrice correspond à deux contraintes scalaires. Ces deux contraintes caractérisent la rotation mais lorsque l'ensemble des variables a un domaine intervalle, ces contraintes ne permettent pas seule d'obtenir la boite minimale, il y a des problèmes de consistance locale. La partie suivante étudie donc l'apport que peut avoir une contrainte globale pour ce problème de rotation.

Amélioration par l'ajout de contrainte globale. Nous connaissons un contracteur optimal $\mathcal{C}^*_{\mathbb{S}}$ pour les contraintes primitives mais le contracteur obtenu grâce à la propagation rétro-propagation n'est pas dans tous les cas optimal. Regrouper certaines contraintes primitives pour les traiter directement de façon optimale permet de réduire ce problème. C'est ce qui est proposé avec l'utilisation de la contrainte angle :

$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = angle\left(\theta, \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}\right). \tag{3.86}$$

Pour cette contrainte, nous n'avons pas encore de contracteur optimal $\mathcal{C}^*_{\mathbb{S}}$.

Sans ce contracteur, l'intérêt d'une telle contrainte peut être mis en évidence par le rajout de contraintes redondantes. Dans une optique de calcul avec les réels, une seule contrainte permet d'obtenir les valeurs souhaitées. Avec l'approche propagation de contraintes sur les intervalles, l'ensemble des contraintes liées à la rotation permettent une contraction plus forte sur les domaines initiaux. Les contraintes supplémentaires qui pénalisent souvent les méthodes classiques permettent ici de relancer la propagation de contraintes.

Pour trouver l'ensemble des contraintes, exprimons la rotation de deux manières différentes. Soit par le déplacement d'un angle θ, soit par un changement de repère comme illustré sur la figure 3-14. Dans le premier formalisme, le point (x_2, y_2) est l'image du point (x_1, y_1) par une rotation dans le sens direct d'angle θ. Dans le second formalisme, (x_1, y_1) et (x_2, y_2) sont les coordonnées

d'un même point dans des repères de même origine mais séparés par une rotation d'un angle θ. Ce qui correspond aux contraintes :

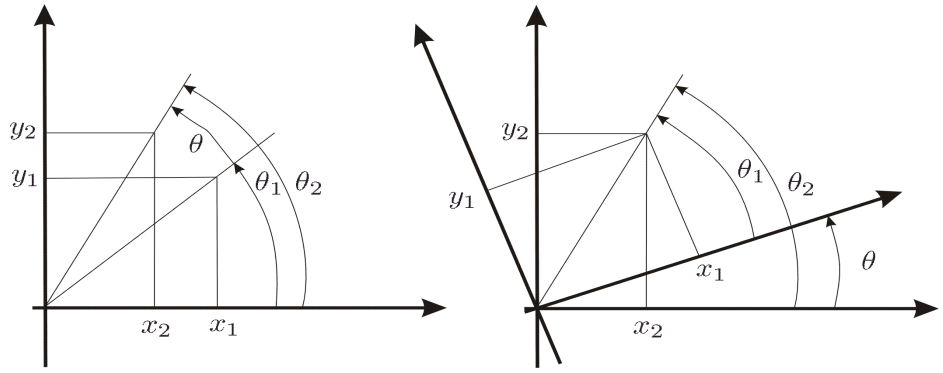

FIG. 3-14 – Correspondance changement de repère et rotation

1) Equations de changement de repère entre le repère 1 et 2 :

$$x_2 = x_1 \cos\theta - y_1 \sin\theta, \tag{3.87}$$
$$y_2 = x_1 \sin\theta + y_1 \cos\theta. \tag{3.88}$$

2) Equations de changement de repère entre le repère 2 et 1 :

$$x_1 = x_2 \cos\theta + y_2 \sin\theta, \tag{3.89}$$
$$y_1 = -x_2 \sin\theta + y_2 \cos\theta. \tag{3.90}$$

3) Egalité des longueurs liant l'origine au point 1 et 2 sur la figure 3-14 :

$$\sqrt{x_1 + y_1} = \sqrt{x_2 + y_2}. \tag{3.91}$$

4) Relation angulaire :

$$\tan(\theta) = \tan(\theta_2 - \theta_1), \tag{3.92}$$
$$= \frac{\sin\theta_2 \cos\theta_1 - \cos\theta_2 \sin\theta_1}{\cos\theta_2 \cos\theta_1 + \sin\theta_2 \sin\theta_1}, \tag{3.93}$$
$$\tan(\theta) = \frac{y_2 x_1 - x_2 y_1}{x_2 x_1 + y_2 y_1}. \tag{3.94}$$

Un test a été réalisé pour illustrer les contractions dues à l'ajout des contraintes redondantes. Sur un pavé initial, tel que

$$[x_1] \times [y_1] \times [\theta] \times [x_2] \times [y_2] = [2,9;3,5] \times [1,9;5,5] \times [-9;0,59632] \times [5,5;5,6] \times [0;10] \tag{3.95}$$

sont appliqués quatre opérateurs de contractions.

Le premier ne prend en compte que les contraintes (3.87) et (3.88), et ne permet (suivant les

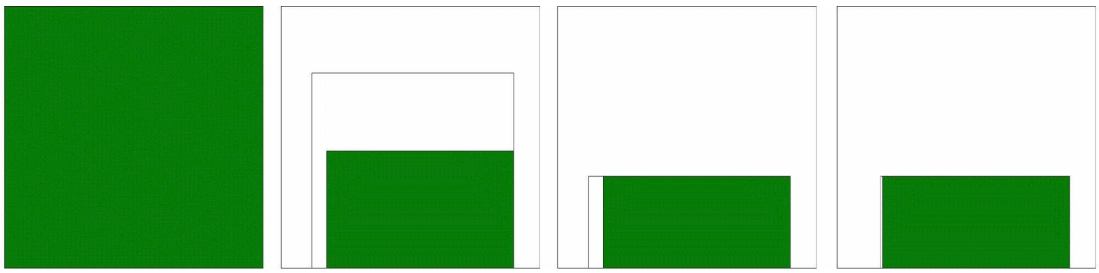

FIG. 3-15 – Illustration des contractions suite à l'ajout de contraintes redondantes, chacune des 4 sous figures correspond au pavé $[-9; 0,59632] \times [5,5; 5,6]$ dans l'espace (θ, x_2).

variables θ et y_2) aucune contraction. Ceci est représenté par le premier carré à gauche sur la figure 3-15.

Les trois carrés suivants correspondent à une contraction due à l'ajout de contraintes redondantes : respectivement de gauche à droite, nous avons ajouté les contraintes (3.89) et (3.90), puis (3.91) et (3.94). Les domaines obtenus sont avec l'ajout de contraintes de plus en plus petits. Ces pavés fixes en gris sur la figure sont obtenus après deux propagations rétro-propagations sur l'ensemble des contraintes primitives et ne sont donc pas obtenus directement.

Remarque 3.2.9 *Le domaine obtenu avec les contraintes redondantes ajoutées n'est pas encore le domaine minimal satisfaisant les contraintes. La mise en place d'un algorithme donnant directement la boîte minimale pour la contrainte globale permettrait d'obtenir ce pavé minimal et de façon plus rapide. Quand une contrainte n'est pas traitée de façon optimale, le pavé fixe est obtenu généralement après plusieures contractions de cette contrainte et le pavé obtenu n'est pas le plus petit. Si le traitement de la contrainte de façon globale nécessite un temps de résolution du même ordre de grandeur que le traitement classique de la contrainte, il y a alors un gain en taille du pavé d'équilibre obtenu mais surtout un gain sur le temps d'obtention de ce pavé car plusieures contractions implique plusieures propagations rétro-propagations sur l'ensemble des contraintes.*

3.3 Bilan

L'analyse par intervalles permet, dans le cas des problèmes de faible dimension, une caractérisation de l'ensemble de solutions. Dans le cadre des problèmes de grande dimension, la caractérisation n'est plus possible.

L'utilisation des méthodes de propagation de contraintes sur les intervalles permet alors d'obtenir des informations sur l'ensemble de solutions sans recourir à la bissection. La méthode de propagation rétro-propagation sur les contraintes que nous utilisons repose sur la création d'opérateurs de projection des contraintes primitives qui ont été présentés dans ce chapitre. Plus

le nombre de contraintes primitives, dont l'opérateur de projection implémenté, est important, plus l'algorithme de propagation peut traiter une large classe de problèmes.

Pour renforcer la contraction obtenue dans le cas des rotations, des contraintes redondantes sont ajoutées au CSP pour simuler une contrainte globale angle, les résultats seront présentés chapitre 5.

Quant au temps de calcul, qui pour les problèmes de grande dimension est aussi un problème déterminant, le chapitre 4 suivant propose une approche pour le réduire. Le codage de la propagation de contraintes ne repose plus sur des arbres représentants les contraintes mais sur un graphe acyclique orienté.

Chapitre 4

Graphe acyclique orienté DAG

Avec le problème d'étalonnage et dans l'optique des CSP de grandes dimensions, le temps de calcul devient rapidement un facteur dont il faut tenir compte. Après avoir effectué un codage des méthodes par propagation sur les intervalles avec des arbres de contraintes, deux options sont possibles pour améliorer le temps de calcul : les méthodes de réveil de contraintes et le codage sous forme de graphe acyclique orienté (Directed Acyclic Graph).

Le DAG n'est pas un outil spécifique aux méthodes par intervalles (voir [4]). Les méthodes de création de DAG sont ainsi largement documentées en particulier en informatique pour la réalisation de compilateurs. Nous présenterons donc ces méthodes dans l'optique d'expliquer le fonctionnement des DAG et leur lien avec les contraintes. La liaison entre contraintes et DAG pour réaliser les algorithmes de propagation rétro-propagation permet donc d'optimiser le traitement des CSP. Ceci ne résout pas les problèmes de convergence lente (algorithme de complexité pseudo-polynomiale) ni les blocages sur des pavés fixes mais permet d'optimiser le temps de calcul, ce qui n'est pas négligeable en particulier pour les CSP de grandes dimensions.

Notre utilisation des DAG pour la propagation de contraintes a été publiée dans JESA [9], depuis d'autres chercheurs ont détaillé leur propre approche en particulier pour l'optimisation globale (voir [64])

4.1 Définitions

Pour introduire les graphes acycliques orientés (que nous appellerons DAG pour Directed Acyclic Graph), présentons quelques définitions de la théorie des graphes [24].

Soit un *graphe* $\mathcal{G} = \{\mathcal{S}, \mathcal{A}\}$

$\mathcal{S}$ = ensemble des sommets du graphe,
$\mathcal{A}$ = ensemble des arcs du graphe.

Un graphe est dit *orienté* quand les arcs ont un sommet initial (origine) et un sommet terminal (destination).

Dans un graphe non orienté les arcs ou *arêtes* n'ont ni sommet origine, ni sommet destination mais deux extrémités.

Exemple 4.1.1 *Soit le graphe orienté* $\mathcal{G}_1 = \{\mathcal{S}_1, \mathcal{A}_1\}$ *avec :*

$$\mathcal{S}_1 = \{s_1, s_2, s_3, s_4\},$$
$$\mathcal{A}_1 = \{a_1, a_2, a_3, a_4, a_5, a_6\}.$$

Sur la figure 4-1, l'arc a_1 *relie le sommet initial* s_1 *au sommet final* s_2 *et permet un déplacement*

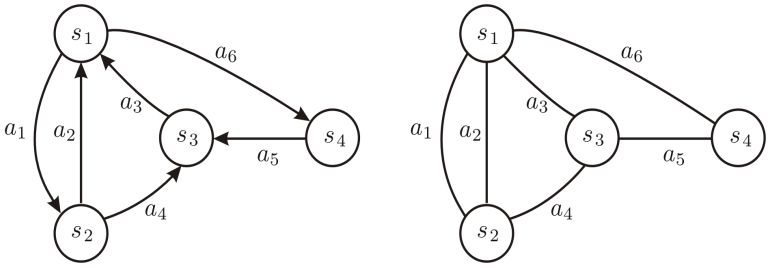

FIG. *4-1 – Représentation d'un graphe orienté et d'un graphe non orienté.*

uniquement de s_1 *vers* s_2 *dans le graphe orienté. Dans le cas du graphe non orienté l'arc* a_1 *est nommé arête et peut être parcouru dans les deux sens.*

La figure 4-2 présente un exemple de chemin (s_2, s_3, s_1, s_4) qui suit la direction des arcs, et de chaîne (s_2, s_3, s_4, s_1) qui, elle ne tient compte que des arêtes. Les définitions sont donc les suivantes :

un *chemin* est une séquence d'arcs orientés consécutifs du graphe,
une *chaîne* est une séquence d'arêtes consécutives.

Grâce à ces deux définitions, nous pouvons introduire les notions de circuit et de cycle suivantes :

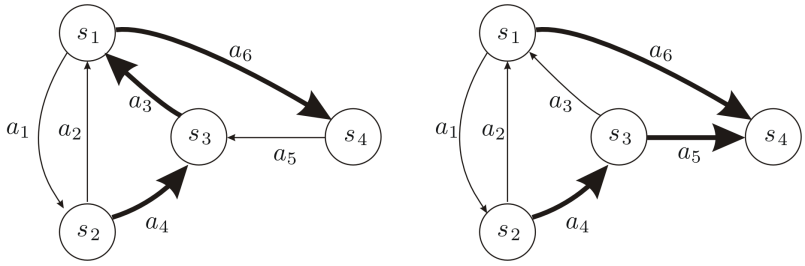

FIG. 4-2 – Illustration des notions de chemin et de chaine.

un *circuit* est un chemin fermé,
un *cycle* est une chaîne fermée.

Sur la figure 4-3, nous pouvons voir le circuit (s_2, s_3, s_1, s_2) et le cycle (s_1, s_3, s_4, s_1).

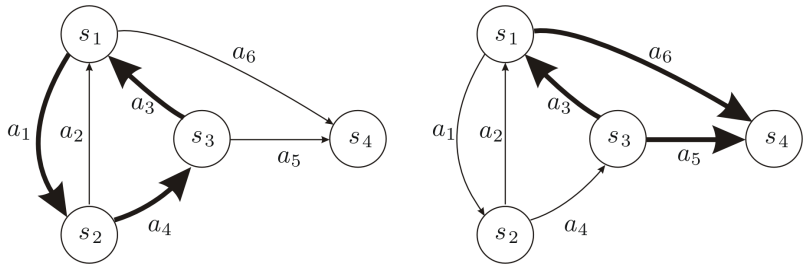

FIG. 4-3 – Illustration des notions de circuit et de cycle.

Pour représenter les contraintes, la première idée est d'utiliser la notion d'arbres d'où les définitions suivantes :

un graphe est *connexe* s'il existe une chaîne entre toute paire de sommets,
un *arbre* est un graphe connexe sans cycle ni circuit.

La deuxième idée pour représenter en mémoire et graphiquement les contraintes n'est plus un arbre mais un DAG.

Un *graphe acyclique orienté DAG* est un graphe orienté ne contenant pas de circuit.

Remarque 4.1.1 *Un DAG peut contenir des cycles. C'est ce point qui différencie un arbre d'un DAG.*

4.2 Comment représenter des contraintes

L'ensemble de ces définitions de la théorie des graphes permettent de comprendre ce que sont les DAG. Reste à illustrer comment ces outils vont nous permettre de représenter les contraintes.

Prenons l'exemple suivant :

$$\begin{aligned} y_1 &= \sin((x_1 + x_2).x_2).\sin(x_1) \\ y_2 &= \max((x_1 + x_2).x_2, x_2.(x_2 + x_3)) \\ y_3 &= \sin((x_2 + x_3).x_2) \end{aligned} \tag{4.1}$$

Ces trois contraintes peuvent se mettre sous la forme de trois arbres comme l'illustre la figure 4-4. Pour chaque arbre, chaque noeud qui n'est pas feuille correspond à une contrainte primitive.

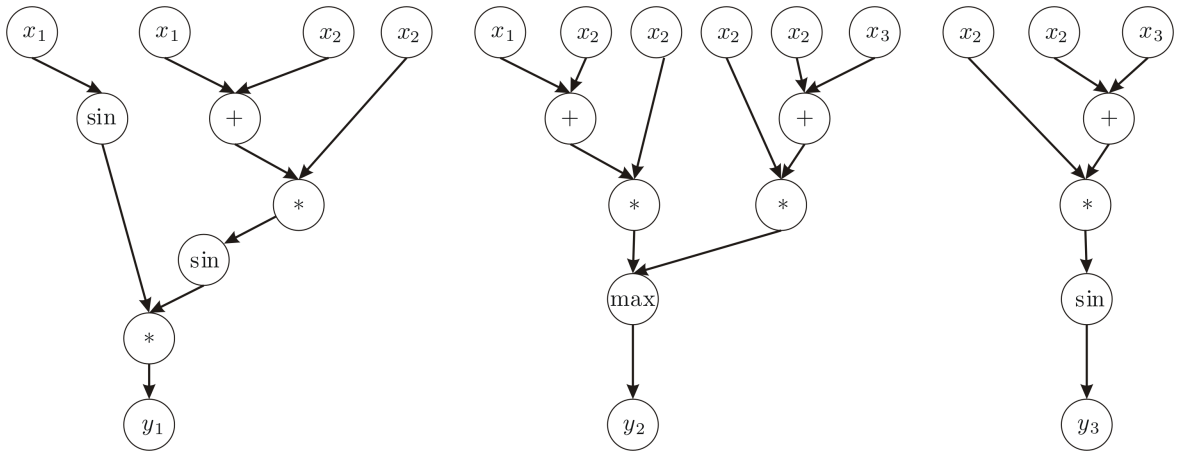

FIG. 4-4 – Représentation des contraintes sous forme d'arbres

Nous obtenons donc la décomposition suivante :

$$\begin{array}{lll} a_1 = x_1 + x_2 & a_5 = x_2 + x_3 & \\ a_2 = a_1.x_2 & a_6 = x_2.a_5 & a_9 = x_2 + x_3 \\ a_3 = \sin(a_2) & a_7 = x_1 + x_2 & a_{10} = a_9.x_2 \\ a_4 = \sin(x_1) & a_8 = a_7.x_2 & y_3 = \sin(a_{10}) \\ y_1 = a_3.a_4 & y_2 = \max(a_6, a_8) & \end{array}$$

Nous avons vu dans le chapitre 3 que c'est sur cette décomposition que se basent les algorithmes de propagations de contraintes. Dans cette décomposition, les variables intermédiaires a_i ne contiennent pas d'information et ont pour domaine initial $]-\infty;+\infty[$.

La mise en place du DAG permet de supprimer des redondances inutiles. Les variables a_1 et a_7 correspondant aux contraintes $a_1 = x_1 + x_2$ et $a_7 = x_1 + x_2$, contiennent le même domaine $[x_1] + [x_2]$ lors de la propagation. Après la rétro-propagation, les domaines sont différents mais seront à nouveau les mêmes à la propagation suivante. La création du DAG permet ainsi de rassembler ces deux contraintes primitives en une seule et ainsi de gagner en temps de calcul.

Pour les contraintes (4.1), nous obtenons figure 4-5 une représentation graphique du DAG. Cette représentation correspond à la décomposition en contraintes primitives suivantes :

$$\begin{array}{lll} b_1 = x_1 + x_2 & b_5 = x_2 + x_3 & \\ b_2 = b_1.x_2 & b_6 = x_2.b_5 & \\ b_3 = \sin(b_2) & & \\ b_4 = \sin(x_1) & & y_3 = \sin(b_6) \\ y_1 = b_3.b_4 & y_2 = \max(b_6, b_2) & \end{array}$$

Dans cette décomposition, les contraintes primitives redondantes ont été supprimées. Les variables intermédiaires b_i jouent alors un rôle identique aux variables a_i précédemment utilisées.

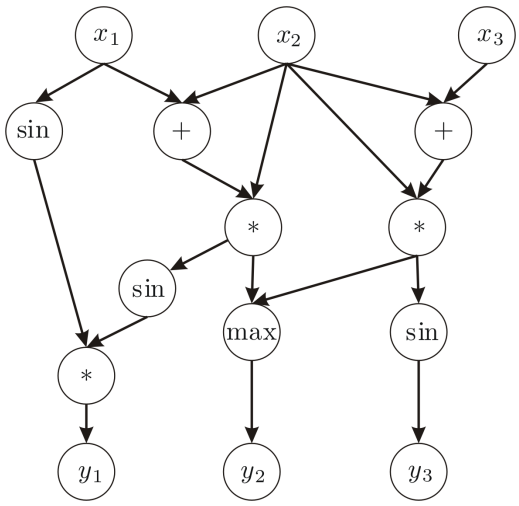

FIG. 4-5 – Représentation des contraintes sous forme de DAG.

4.3 Création du DAG

Nous venons de voir comment représenter les contraintes sous forme de DAG. Avant de détailler la propagation sur le DAG dans la section 4.4, nous proposons de comprendre la création du DAG. Cette partie aborde la partie pratique de la création mais aussi l'utilisation des DAG, en particulier le formalisme entrée-sortie qui est lié aux problèmes que nous traitons.

4.3.1 Une structure entrée-sortie

La structure sous-jascente au DAG est une structure entrées-sorties avec pour variables d'entrées les variables sources de notre problème, et pour variables de sorties les variables qui peuvent être calculées à partir de ces variables d'entrées. Pour illustrer cette structure, prenons les contraintes suivantes :

$$\begin{aligned} y_1 &= x_2.(x_2 + x_1) \\ y_2 &= x_2.(x_2 + x_1) + \sin(x_2) \\ y_3 &= \sin(x_2).x_3 \end{aligned}$$

qui se décomposent en contraintes primitives de la façon suivante :

$$\begin{array}{lll} a_1 = x_2 + x_1 & a_3 = \sin(x_2) & \\ a_2 = x_2.a_1 & a_4 = a_2 + a_3 & a_5 = a_3.x_3 \\ y_1 = a_2 & y_2 = a_4 & y_3 = a_5 \end{array}$$

Chacune des variables x_i, a_i, et y_i représente un noeud du graphe de la figure 4-6 et les arcs représentent le fait que deux variables sont liées par une contrainte primitive. En choisissant

x_1, et x_2 comme variables d'entrée nous pouvons calculer a_1 et a_3. Ces deux variables étant calculables à partir des variables sources, elles peuvent être utilisées pour calculer de nouvelles variables a_1 et a_3. En répétant le raisonnement, nous obtenons les variables de sorties y_1 et y_2. Restent sur ce schéma les variables x_3, a_5 et y_3 qui ne sont pas calculables. En effet la contrainte $a_5 = a_3.x_3$ ne permet pas de calculer x_3 et a_5 uniquement à partir de a_3.

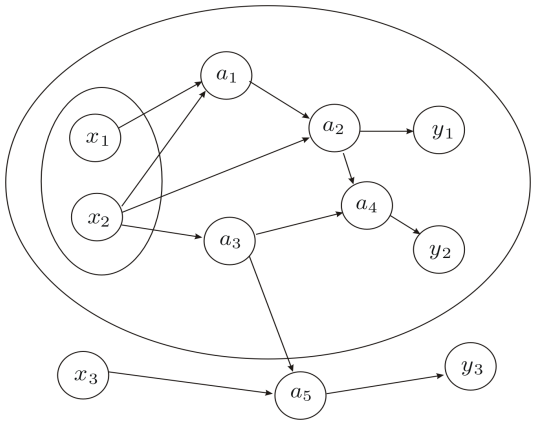

FIG. 4-6 – Illustration des variables d'entrée et de sortie. La petite ellipse représente le choix des variables d'entrée et la grande les variables calculables à partir de ces entrées.

Remarque 4.3.1 *Un autre choix de variables d'entrées est possible, en particulier les trois variables x_i. Ce choix permet au DAG de couvrir l'ensemble des contraintes contrairement au choix précédent.*

Le choix des variables d'entrées revient donc au créateur du DAG et est très lié aux problèmes que nous cherchons à résoudre.

Exemple 4.3.1 *Pour illustrer comment les problèmes d'automatique et de robotique correspondent au formalisme introduit, détaillons les trois problèmes suivants :*
1) Des équations d'état à temps discret :

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{e}(k)) \tag{4.2}$$
$$\mathbf{y}(k) = \mathbf{g}(\mathbf{x}(k), \mathbf{e}(k)) \tag{4.3}$$

où nous pouvons choisir $x(0)$ et $e(k)$ l'erreur de modèle comme variables d'entrées et $y(1), ...$ $y(k_{max})$ pour celles de sorties.
2) Le problème de l'image directe pour les robots parallèles, en particulier l'exemple d'une plate-forme de Gough Stewart. Pour une position et une orientation données et avec les paramètres de la plate-forme, il est possible de calculer les longueurs des vérins actionnant la plate-forme

$$\mathbf{q} = \mathbf{f}(\mathbf{x}, \mathbf{p}).$$

Nous avons $\mathbf{x}$ *et* $\mathbf{p}$ *en entrée,* $\mathbf{q}$ *en sortie.*
3) L'estimation de paramètres s'écrit

$$\mathbf{y} = \mathbf{f}(\mathbf{x}, \mathbf{p})$$

où nous cherchons $\mathbf{p}$ *connaissant* $\mathbf{y}$ *et* $\mathbf{x}$*. Ici les variables d'entrées sont* $\mathbf{p}$ *et* $\mathbf{x}$ *celles de sorties* $\mathbf{y}$*. Dans ces exemples apparaît clairement la différence entre variables de sortie et variables recherchées. En effet les variables de sortie sont celles qui sont "calculables" à partir des variables d'entrée.*

Remarque 4.3.2 *Le DAG dans notre implémentation n'est pas unique. En effet une expression du type :*

$$x_1 + x_2 + x_3$$

pourra être interprétée de trois façons suivantes :

$$\begin{array}{lll} a_1 = x_1 + x_2 & a_1 = x_1 + x_3 & a_1 = x_2 + x_3 \\ a_2 = a_1 + x_3 & a_2 = a_1 + x_2 & a_2 = a_1 + x_1 \end{array}$$

en fonction de l'ordre de passage lors de la création du DAG. Une alternative peut se situer dans la prise en compte non plus de noeuds binaires mais des noeuds n-aires.

4.3.2 Algorithme de création

Pour comprendre la création du DAG, la structure des contraintes primitives sous forme d'arbres et le formalisme entrée-sortie sont nécessaires. Les entrées de l'algorithme sont donc les variables d'entrée et l'ensemble des arbres de contraintes comme dans l'exemple de la figure 4-4. La procédure de création suit le principe d'un algorithme de marquage des noeuds calculables et se décompose de la manière suivante :

1. Création d'une pile avec tous les noeuds des arbres correspondant aux variables d'entrée et aux nombres réels.
2. Tant que la pile n'est pas vide :
   (a) Examen d'un noeud de la pile et de ses descendants. S'ils sont calculables, alors la contrainte primitive est validée.
   (b) Si cette contrainte se situe déjà dans la liste des contraintes du DAG, elle n'est pas ajoutée. Sinon elle s'ajoute aux autres contraintes et les descendants du noeud étudié sont ajoutés à la pile.

La procédure s'arrête avec le vidage de la pile. L'ensemble des variables calculables sont alors présentes dans le DAG. Certaines contraintes primitives et variables peuvent alors ne pas être reconnues, un test final permet de visualiser ces contraintes et donc les variables associées.

Remarque 4.3.3 *Le fait de connaître les variables de sorties de notre DAG, permet de réaliser ce test facilement.*

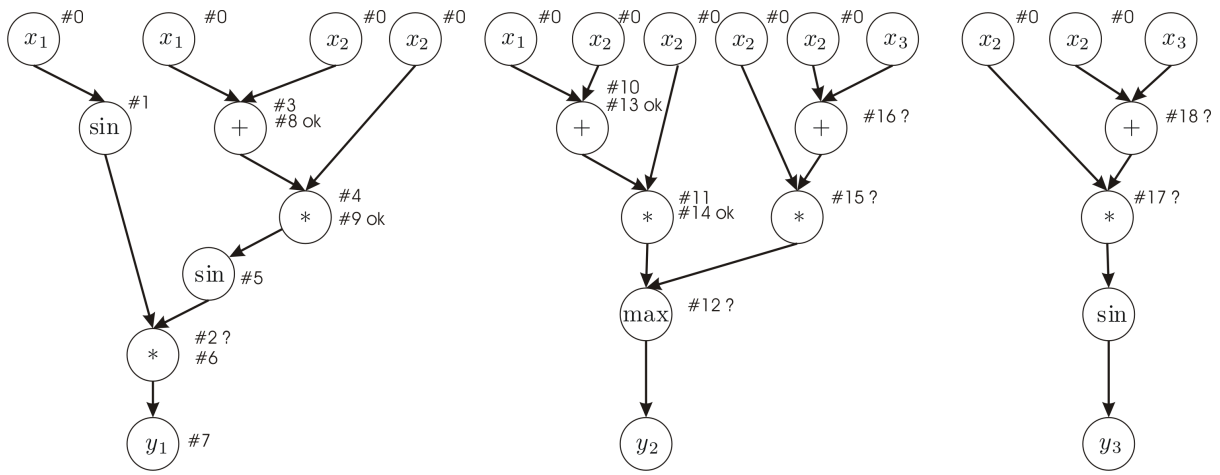

FIG. 4-7 – Figure illustrant la création du DAG avec comme variables d'entrée x_1 et x_2.

Sur l'exemple 4.1, si pour la création nous choisissons x_1 et x_2 comme variables d'entrée, la procédure de création du DAG se représente comme sur la figure 4-7. Le marquage #0 correspond à la création de la pile des variables d'entrée. Ensuite chaque $\#i$ correspond à l'étude du noeud marqué. S'il est simplement marqué $\#i$, la contrainte primitive le liant à ces prédécesseurs vient d'être intégrée au DAG. S'il est suivi de "*ok*" ou de "?", le noeud a alors soit déjà été traité soit ne peut pas être calculé à partir de ses prédécesseurs à l'étape i du programme.

Le changement de variables d'entrée permet de visualiser l'arrêt du programme dans un cas où toutes les contraintes ne peuvent pas être reconnues. Ceci est visualisé sur la figure par l'absence de marquage sur certains noeuds.

4.3.3 Question complexité

La complexité de l'algorithme de création du DAG est liée au nombre de contraintes primitives présentes dans l'ensemble des arbres représentant les contraintes. En effet il faut tester l'ensemble des noeuds pour savoir s'ils peuvent être marqués.

Lors de la création du DAG, il faut très rapidement savoir si une contrainte primitive appartient au DAG. Une première solution consiste donc à parcourir la liste des contraintes primitives du DAG. Cette solution nécessite, pour trouver les éléments de la liste, 1 itération pour le premier élément, 2 pour le deuxième,... Nous avons donc une suite $1 + 2 + 3 + ... + n$ qui correspond à $(n + 1).n/2$ comparaisons de chaînes de caractères.

L'autre solution consiste en la réalisation d'une table de hachage de dimension m. Cette table permet de découper la liste des chaînes de caractères en m listes rangées dans un tableau comme

l'illustre la figure 4-8. Pour tester la présence d'une chaîne a, le principe de fonctionnement est

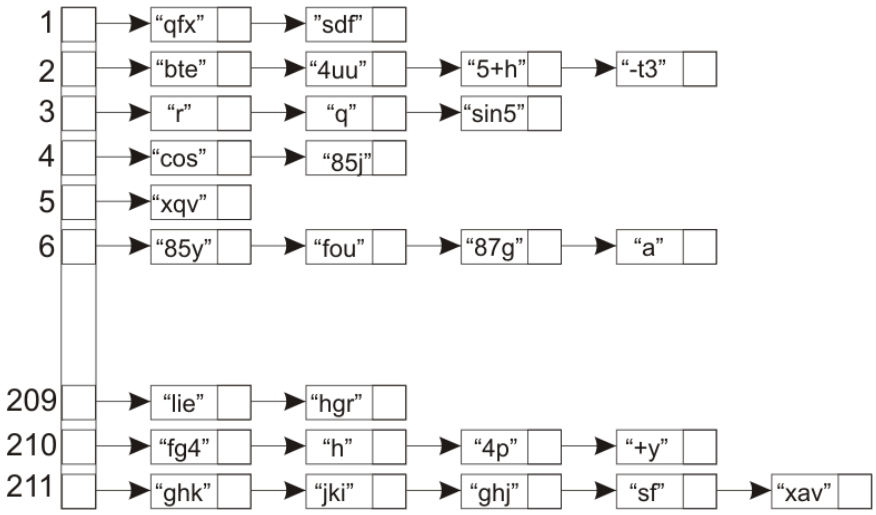

FIG. 4-8 – Illustration du principe de la table de hachage.

le suivant :

1. Une fonction de hachage associe à chaque chaîne de caractères un nombre entre 1 et m

$$f : \mathbb{CH} \rightarrow \{1, 2, \dots, m\}, \tag{4.4}$$

   où $\mathbb{CH}$ est l'ensemble des chaînes de caractères. Pour a, associons le nombre i.
2. Si la chaîne de caractères appartient à la liste stockée dans la $i$ème ligne du tableau, nous avons vérifié la présence de a dans la table. Sinon a est inséré en fin de cette chaîne et dans notre cas dans le DAG.

Chaque liste du tableau ayant moins d'éléments, la table de hachage permet approximativement de diviser par m le nombre de tests de chaînes de caractères pour retrouver un élément.

Remarque 4.3.4 *Pour que la table fonctionne de façon optimale, il faut que la fonction de hachage répartisse de façon homogène les chaînes de caractères. L'idéal étant que la table soit rectangulaire : moins d'éléments classés dans chaque ligne de la table que de lignes. Pour des exemples de table de hachage, il est possible de consulter [4].*

4.4 Application des DAG aux méthodes par intervalles

Nous avons vu le principe de la propagation rétro-propagation dans le chapitre 3 et comment l'élimination de contraintes redondantes permettait l'amélioration de la structure des contraintes sous forme de DAG (partie 4.2). Pour détailler le fonctionnement de la propagation de contraintes sur les DAG, reprenons l'exemple 4.1 pour chacune des étapes de la méthode.

4.4.1 Propagation dans le DAG

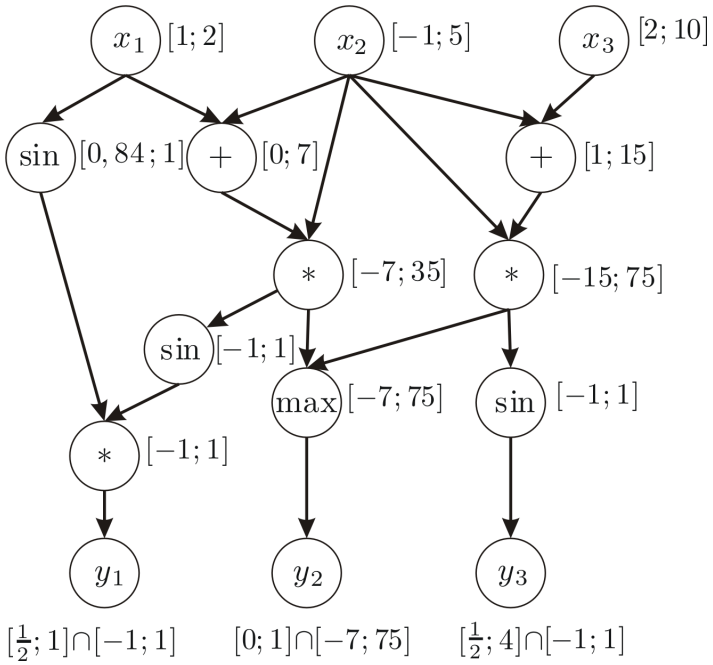

FIG. 4-9 – Propagation sur les DAG

Les variables d'entrée x_i ont pour domaines associés $[x_i]$. A partir de ces domaines les contraintes primitives peuvent être évaluées par intervalles, nous avons donc par exemple $\sin(x_1) \in [0, 84; 1]$, $x_1 + x_2 \in [0; 7]$, ... Sur la figure 4-9, l'ensemble des domaines intervalles associés à chaque noeud du DAG est représenté après le premier passage de propagation. L'intersection entre le domaine d'appartenance des variables y_i et le domaine obtenu avec le DAG permet d'obtenir le domaine $[y_i]$ pour la première propagation.

Chaque noeud ne portant pas d'information lors de la première propagation, nous pouvons aussi représenter le domaine intervalle obtenu pour chaque noeud comme l'intersection avec l'intervalle $]-\infty; +\infty[$. Après rétro-propagation comme nous allons le voir, l'ensemble des domaines peut être modifié. Les propagations s'effectueront donc grâce à l'intersection du domaine obtenu par la rétro-propagation et du domaine obtenu par évaluation du noeud.

4.4.2 Rétro-propagation

La propagation permet de faire passer l'information contenue dans les domaines associés aux variables x_i vers les variables y_i, elle supprime des valeurs du domaine $[y_i]$ ne vérifiant pas la contrainte quand $x_i \in [x_i]$. La rétro-propagation fait l'inverse et supprime en tenant compte des domaines $[y_i]$, des valeurs non consistantes sur les domaines de chaque noeud du DAG.

Sur la figure 4-11, le noeud correspondant à la contrainte primitive $y_1 = b_3.b_4$, noeud en gras

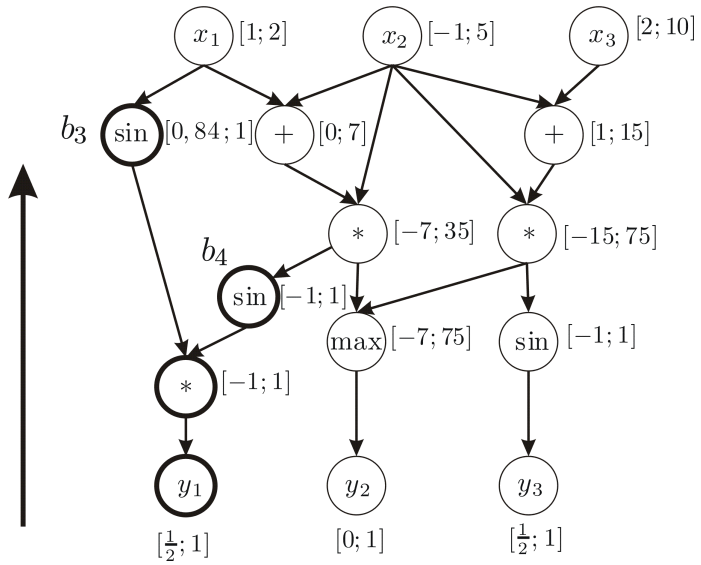

FIG. 4-10 – Présentation de la rétro-propagation sur le noeud $y_1 = b_3.b_4$ en gras sur la figure.

sur la figure 4-10, permet de réduire le domaine $[b_3]$. C'est la connaissance conjointe de trois domaines $[y_1]$, $[b_3]$ et $[b_4]$ qui a permis cette contraction, comme l'illustre la figure 4-11. Pour chacun des noeuds du DAG, la rétro-propagation va être effectuée jusqu'aux domaines pour les variables x_i.

Le processus de propagation rétro-propagation va donc permettre de réduire tour à tour l'ensemble des domaines intervalles jusqu'à l'obtention d'un pavé d'équilibre. Le nombre d'étapes de propagation rétro-propagation n'est pas fixé, tant que les domaines peuvent être contractés, le processus peut continuer.

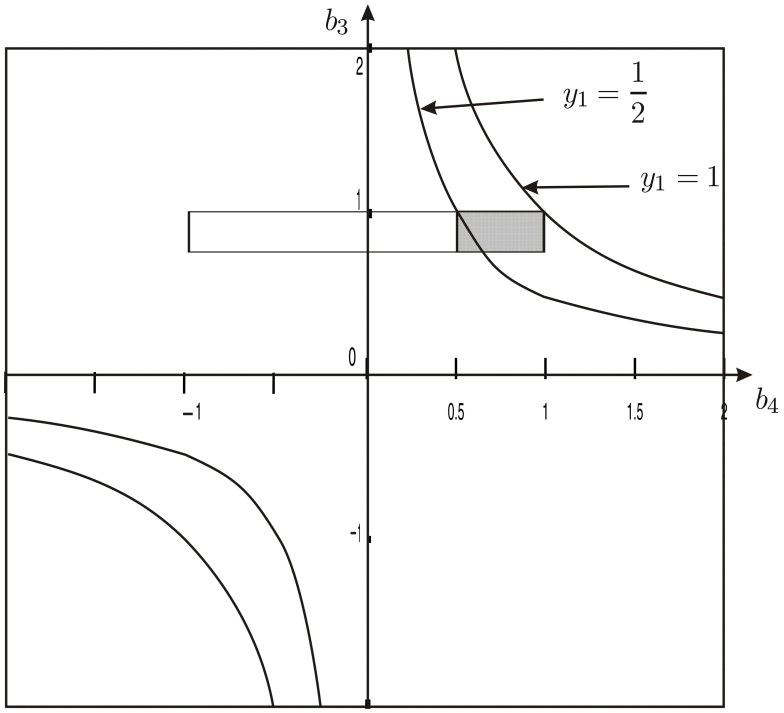

FIG. 4-11 – Illustration de la contraction d'un noeud lors de la rétro-propagation.

Chapitre 5

Applications

Ce chapitre présente la partie plus applicative de la thèse. En effet, cette thèse a permis la réalisation d'un logiciel IntervalPeeler qui met en place les techniques de propagation de contraintes pour l'étalonnage des robots mais aussi pour de nombreux autres problèmes. Le choix que nous avons pris est de ne jamais effectuer de bissection pour ne pas entrer dans le cadre des algorithmes de complexité exponentielle. IntervalPeeler est donc conçu dans l'optique de rechercher la plus petite boîte possible encadrant un ensemble le plus rapidement possible. Le nom d'IntervalPeeler a donc été choisi pour l'analogie avec le couteau éplucheur, le principe du solveur étant de supprimer les parties non consistantes des domaines intervalles initiaux vis-à-vis des contraintes.

D'autres solveurs existent tels que SpaceSolver [52], [3], AQCS [63] (Approximate Quantified Constraint Solving) ou RealPaver [31], [30] ou des librairies en C++ comme Boost et Alias [56] et posent alors la question de l'utilité de la mise en place d'un nouveau solveur. Les raisons qui nous ont orientés vers ce choix sont les suivantes :

1. Au début de la thèse, certains de ces solveurs n'existaient pas ou ne prenaient pas en compte l'ensemble des contraintes dont nous avions besoin (contraintes trigonométriques par exemple)
2. Pour comprendre les méthodes mises en jeu et pour situer où les points de blocage apparaissent dans les problèmes, la mise en oeuvre des algorithmes permet de clarifier certains points. Ils permettent alors à notre intuition de se développer conformément à la méthode avec la possibilité de coder de nouveaux opérateurs de contraction telle que la contrainte angle (voir page 64).
3. Le choix de la mise en oeuvre des DAG (voir page 69) est alors possible et permet une comparaison avec les autres méthodes.
4. Enfin IntervalPeeler permet un accès aux méthodes par contraintes dans un but pédagogique pour tester rapidement des petits problèmes.

Bien sûr, le choix de la réalisation présente aussi des inconvénients comme :

1. Une absence de codage de fonctionnalités importantes (arrondi extérieur, infini).

2. Une performance moindre sur certains problèmes (problèmes formalisés avec des contraintes qui ne sont pas sous la forme $\mathbf{y} = \mathbf{f}(\mathbf{x})$).

3. Une nécessité pour l'utilisateur de donner des variables d'entrée et de sortie pour son problème et donc de connaître un peu le formalisme des DAG employés dans IntervalPeeler.

IntervalPeeler a donc été réalisé pour un usage pédagogique et pour un usage de recherche. Nous retrouverons donc ces deux fonctionnalités avec un mode expert pour l'utilisation des DAG et un mode Utilisateur simple avec interface graphique.

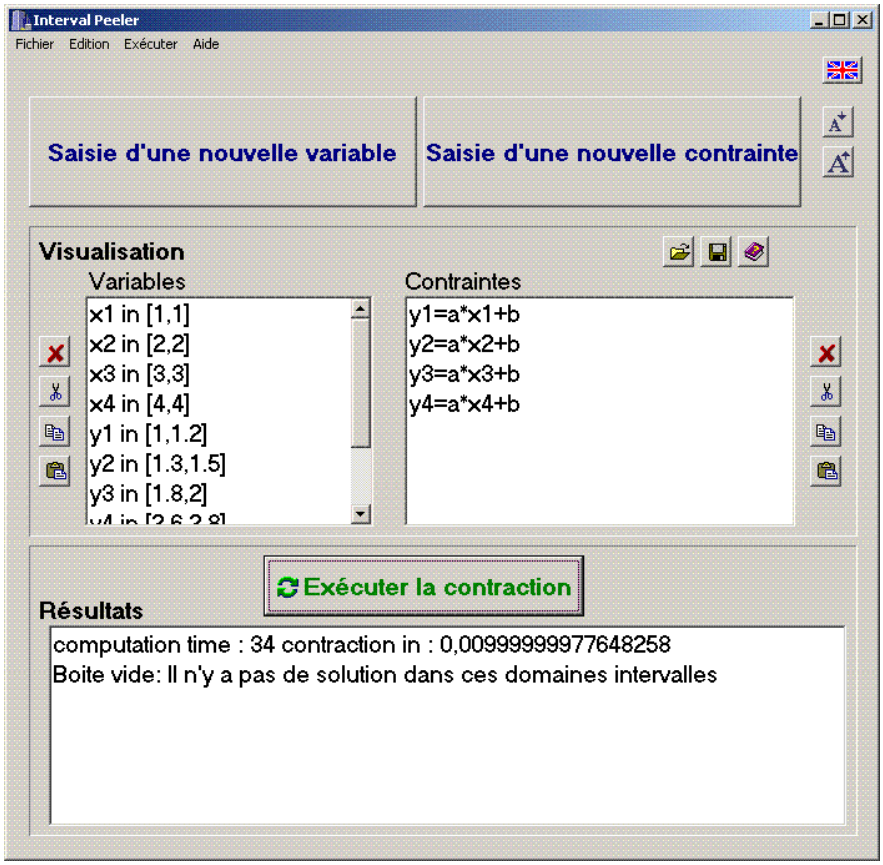

Fig. 5-1 – Interface graphique d'IntervalPeeler

Avec ce logiciel, plusieurs types de problèmes peuvent être traités : l'estimation de paramètres, l'estimation d'état, la commande robuste, la détection de défauts et, bien sûr, l'étalonnage des robots. Ces applications issues de différents articles donnent une idée de la classe de problèmes que peut traiter IntervalPeeler. Elles sont donc présentées dans ce chapitre avant de détailler les résultats obtenus pour l'étalonnage des robots STAUBLI rx90.

5.1 IntervalPeeler

La première utilisation possible d'IntervalPeeler est un usage à but pédagogique pour traiter de petits exemples avec l'interface graphique. Cette interface visualisée sur la figure 5-1 affiche la liste des variables ainsi que les domaines associés, la liste des contraintes du CSP et les domaines obtenus dans l'espace **Résultats**.

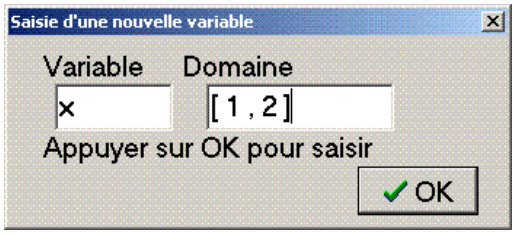

Fig. 5-2 – Fenêtre de saisie d'une variable et de son domaine.

Les variables sont saisies grâce au bouton **Saisie d'une nouvelle variable** et doivent suivre la syntaxe suivante :

a) Pour saisir des variables, il faut que la variable commence par une lettre (A-Za-z) puis utilise toutes les combinaisons de lettres et de chiffres, par exemple :

$$A1, A456a, test1, variable4... \tag{5.1}$$

b) Les domaines intervalles associés à ces variables étant définis par la syntaxe suivante :

$$[borne\_inférieure; borne\_supérieure] \text{ soit par exemple } [1; 2]. \tag{5.2}$$

La fenêtre de saisie est présentée figure 5-2.

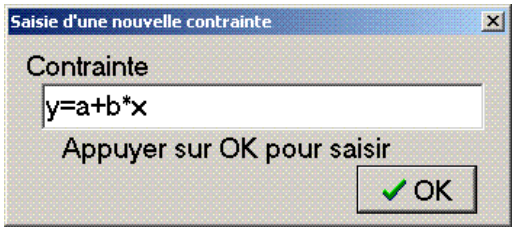

Fig. 5-3 – Fenêtre de saisie d'une contrainte.

Pour saisir les contraintes, le bouton **Saisie d'une nouvelle contrainte** de la figure 5-1 fait apparaître une fenêtre de saisie similaire (voir figure 5-3) qui permet de saisir l'ensemble des contraintes ou des combinaisons de contraintes primitives du tableau **Syntaxe d'**Interval Peeler.

Contraintes Arithmétiques	Contraintes Puissances	Contraintes MinMax
$a = b + c$	$a = \text{sqr}(b)$	$a = \max(b, c, d, ...)$
$a = b - c$	$a = \text{sqrt}(b)$	$a = \min(b, c, d, ...)$
$a = b * c$	$a = b\hat{}n\ ,\ n \in \mathbb{N}$	
$a = b/c$		Autres
Contraintes trigonométriques		$a = \text{abs}(b)$
$a = \sin(b)$	Contraintes exponentielles	$a = \text{step}(b)$
$a = \cos(b)$	$a = \exp(b)$	$\text{boolean}(b)$
$a = \tan(b)$	$a = \ln(b)$	$\text{integer}(b)$

Tableau 5.0 : Syntaxe d'IntervalPeeler

La deuxième utilisation d'IntervalPeeler est l'utilisation en mode expert (Exécuter->Expert + Edition->Utiliser l'éditeur Notepad) avec l'utilisation d'un fichier texte en entrée qui suit la syntaxe suivante :

```
Input Variables
   x1 in [1,2]
   x2 in [3,4]
   x3 in [5,6]
Output Variables
   y1 in [-oo,+oo]
   y2 in [-oo,+oo]
   y3 in [-oo,+oo]
Constraints
   y1=sin(x1)/x1
   y2=sin(x2)/x2
   y3=sin(x3)/x3
   //contrainte redondante
   y1=100*y3*y2
End
```

Lors de l'utilisation de cette option, l'usage des DAG est optimale. Le choix des variables d'entrée et de sortie est alors essentiel (voir chapitre 4). Le temps de calcul pouvant varier en fonction des différents choix possibles. Les choix impossibles, par exemple une variable de sortie non calculable à partir des variables d'entrée, sont indiqués par un message d'erreur qui suit la compilation des contraintes.

5.2 Estimation de paramètres

Les problèmes d'estimation de paramètres peuvent, comme pour l'étalonnage des robots, se formaliser sous la forme de CSP. La recherche de paramètres pouvant être effectuée pour de nombreux problèmes d'automatique, de mécanique,... ou comme pour l'exemple suivant d'électronique :

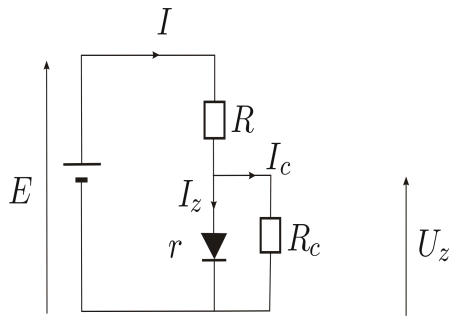

FIG. 5-4 – Présentation du circuit électrique

Le circuit électrique de la figure 5-4 est composé d'un générateur continu délivrant une tension E, de deux résistances R et R_c et d'une diode. Les lois de l'électrocinétique et les caractéristiques de la diode nous permettent d'écrire les 4 relations suivantes :

$$\begin{array}{ll} \text{i) } I_z = \max(0, \frac{U_z - U_0}{r}), & \text{ii) } I = I_c + I_z, \\ \text{iii) } U_z = R_c.I_c, & \text{iv) } E = R.I + U_z. \end{array} \tag{5.3}$$

Par ailleurs, les mesures effectuées sur l'intensité et les tensions sur le circuit avec une certaine incertitude, permettent d'affecter aux variables des domaines d'appartenance tels que :

$$U_z \in [6V\ ; 7V\ ],\ I \in [0,11; 0,12]. \tag{5.4}$$

Pour le reste des variables, les domaines sont donnés grâce aux caractéristiques des éléments présents dans le circuit. Nous avons donc l'ensemble des solutions pour notre problème :

$$\mathbb{S} = \left\{ \begin{pmatrix} r \\ U_0 \\ R \\ E \\ I_z \\ U_z \\ I \\ I_c \\ R_c \end{pmatrix} \in \begin{pmatrix} [7;8] \\ [6;6,2] \\ [100\ ;110] \\ [18;20] \\ [0,01\ ;+\infty\ [\\ [6\ ;7\ ] \\ [0,11;0,12] \\ ]-\infty;+\infty[\\ [50;60] \end{pmatrix} \text{ tels que } \begin{cases} I_z = \max(0, \frac{U_z - U_0}{r}) \\ U_z = R_c.I_c \\ I = I_c + I_z \\ E = R.I + U_z \end{cases} \right\},$$

Ce CSP permet donc de générer pour IntervalPeeler le fichier source suivant :

```
Input variables
   r in [7,8]
   U0 in [6,6.2]
   R in [100 ,110]
   E in [18,20]
   Iz in [0.01 ,+oo ]
   Uz in [6 ,7 ]
   I in [0.11,0.12]
   Ic in [  -oo ,+oo   ]
   Rc in [ 50  ,60   ]
Output variables
Constraints
   Iz=max(0,(Uz-U0)/r)
   Uz=Rc*Ic
   I=Ic+Iz
   E=R*I+Uz
End
```

La résolution de ce CSP avec IntervalPeeler génère une seule boîte solution dont nous sommes sûrs qu'elle enferme $\mathbb{S}$. Ce résultat se traduit par les relations d'appartenance :

$$\begin{array}{ll} r \in [7;8], & I_z \in [0,01;0,03813], \\ U_0 \in [6;6,2], & U_z \in [6,07;6,3507], \\ R \in [100;110], & I \in [0,11117;0,12], \\ E \in [18;19,5507], & I_c \in [0,10117;0,11], \\ R_c \in [55,18;60]. & \end{array} \tag{5.5}$$

Dans ces résultats, non seulement R_c et E voient leurs domaines se contracter, mais aussi les mesures I et U_z, ceci traduit le fait que certaines valeurs des domaines étaient inconsistantes et n'avaient donc pas de réalité physique.

Remarque 5.2.1 *Cet exemple peut être traité par* RealPaver *[31] qui donne les mêmes résultats, un exemple similaire a été publié dans [37].*

5.3 Commande robuste

Le problème de la commande robuste est de garantir la stabilité d'un système même s'il existe une variation des paramètres du système ou du contrôleur lui-même. C'est un problème difficile car il faut alors prouver qu'un ensemble de solutions sont vérifiées. Les méthodes utilisant l'analyse par intervalles sont donc particulièrement adaptées au problème de la commande robuste [49, 41, 47, 48, 54].

Pour illustrer ce problème, prenons un deuxième exemple de [37]. Soit l'équation différentielle :

$$a_n y^{(n)} + a_{n-1} y^{(n-1)} + \cdots + a_1 \dot{y} + a_0 y = b_m u^{(m)} + a_{m-1} u^{(m-1)} + \cdots + b_1 \dot{u} + b_0 u, \tag{5.6}$$

qui régit la dynamique de ce système où y est la sortie et u l'entrée. La relation entrée-sortie après l'application de la transformée de Laplace est donc :

$$y(s) = \frac{b_m s^m + \cdots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0} u(s). \tag{5.7}$$

Le polynôme du dénominateur :

$$P(s) = a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0 \tag{5.8}$$

est appelé *polynôme caractéristique.* C'est lui qui détermine le comportement dynamique du système. Par exemple, si toutes ses racines sont à parties réelles négatives, alors le système est stable, c'est-à-dire que si l'entrée $u(t)$ du système passe soudainement à zéro, alors la sortie va converger vers zéro. Le critère de Routh que nous allons maintenant présenter (voir [25] page 278 pour plus de détails) permet de prouver la stabilité d'un système avec un nombre d'opérations très faible, sans avoir à calculer toutes les racines du polynôme caractéristique.

Théorème 5.1 (Critère de Routh). Soit $P(s) = a_n s^n + \cdots + a_1 s + a_0$ un polynôme. La table de Routh se construit comme suit :

a_n	a_{n-2}	a_{n-4}	a_{n-6}	$\ldots$	0	0
a_{n-1}	a_{n-3}	a_{n-5}	a_{n-7}	$\ldots$	0	0
b_1	b_2	b_3			0	0
c_1	c_2	c_3			0	0
$\vdots$	$\vdots$	$\vdots$			$\vdots$	$\vdots$

(5.9)

avec :

$$\begin{array}{lll} b_1 = \frac{a_{n-1}a_{n-2} - a_n a_{n-3}}{a_{n-1}} & b_2 = \frac{a_{n-1}a_{n-4} - a_n a_{n-5}}{a_{n-1}} & \cdots \\ c_1 = \frac{b_1 a_{n-3} - a_{n-1} b_2}{b_1} & c_2 = \frac{b_1 a_{n-5} - a_{n-1} b_3}{b_1} & \cdots \\ \cdots & \vdots & \end{array} \tag{5.10}$$

Notons que les deux premières lignes sont les coefficients de $P(s)$. Les autres éléments t_{ij} à la

ligne $i > 2$ et à la colonne j sont obtenus par le calcul :

$$t_{ij} = \frac{t_{i-1,1}t_{i-2,j+1} - t_{i-2,1}t_{i-1,j+1}}{t_{i-1,1}}. \tag{5.11}$$

Les racines de $P(s)$ sont toutes à parties réelles strictement négatives si et seulement si les éléments de la première colonne de la table de Routh sont tous de même signe.

Considérons à titre d'exemple le problème d'une moto roulant à une vitesse constante et faible de 1m/s (voir [25] page 313 pour plus de détails sur ce problème). L'entrée du système est l'angle θ du guidon et la sortie est l'angle de roulis ϕ de la moto. La relation entrée-sortie du système est :

$$\phi(s) = \frac{1}{s^2 - \alpha_1}\theta(s) \tag{5.12}$$

Vu la très faible vitesse de la moto, l'effet gyroscopique de la roue avant (qui maintient la moto stable à vive allure) devient négligeable et le système est donc instable. Nous proposons de commander l'angle du guidon de la moto par un régulateur du premier ordre dont la relation entrée-sortie, donnée dans le domaine de Laplace, est

$$\theta(s) = \frac{\alpha_2 + \alpha_3 s}{\tau s + 1}\left(\phi_d(s) - \phi_m(s)\right) \tag{5.13}$$

où ϕ_d est l'angle de roulis désiré et ϕ_m est l'angle de roulis mesuré. Notons que comme le capteur n'est pas parfait, l'angle mesuré ϕ_m, c'est-à-dire celui qui apparaît en sortie du capteur, n'est pas exactement l'angle réel ϕ de la moto. La relation différentielle entre ces deux quantités est donnée par :

$$\phi_m(s) = \left(1 + 2s + ks^2\right)\phi(s) \tag{5.14}$$

L'ensemble de ces équations est représenté par le schéma de la figure 5-5. En manipulant les trois

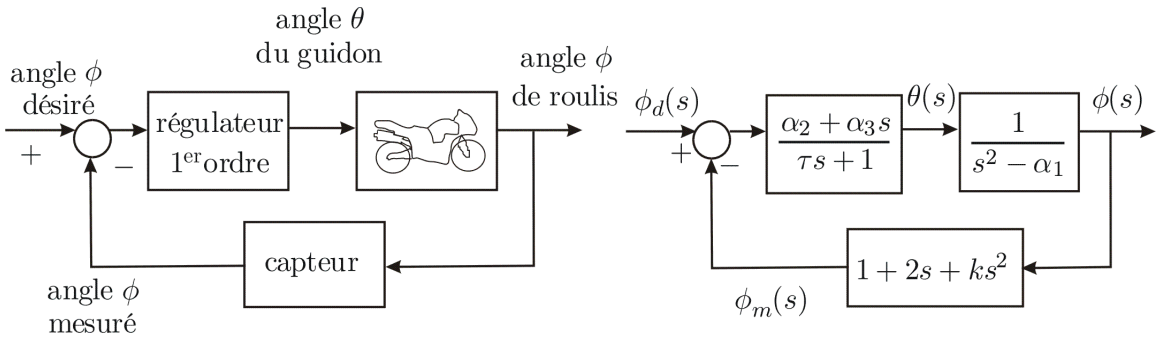

FIG. 5-5 – Schéma du modèle de la moto et de son contrôleur.

équations précédentes de façon à supprimer $\theta(s)$ et $\phi_m(s)$, l'équation entrée-sortie du système bouclé devient :

$$\phi(s) = \frac{\alpha_2 + \alpha_3 s}{(s^2 - \alpha_1)(\tau s + 1) + (\alpha_2 + \alpha_3 s)(1 + 2s + ks^2)}\phi_d(s)$$

Son polynôme caractéristique est :

$$\left(s^2-\alpha_1\right)(\tau s+1)+(\alpha_2+\alpha_3 s)\left(1+2s+ks^2\right)=a_3s^3+a_2s^2+a_1s+a_0$$

avec $a_3=\tau+\alpha_3 k$, $a_2=\alpha_2 k+2\alpha_3+1$, $a_1=\alpha_3-\alpha_1\tau+2\alpha_2$ et $a_0=-\alpha_1+\alpha_2$. La table de Routh associée est donc :

a_3	a_1
a_2	a_0
$\frac{a_2a_1-a_3a_0}{a_2}$	0
a_0	0

(5.15)

Ainsi, le système est stable si $a_3, a_2, \frac{a_2a_1-a_3a_0}{a_2}$ et a_0 sont de même signe.

Les valeurs nominales pour les coefficients sont : $\alpha_1=9, \alpha_2=3, \alpha_3=1, \tau=2, k=-3$. Or, ces coefficients ne sont connus que de façon approximative. En pratique, nous connaissons un intervalle d'appartenance pour chacune de ces variables. Notons les $[\alpha_1], [\alpha_2], [\alpha_3], [\tau]$ et $[k]$. Le problème de stabilité robuste consiste à montrer la stabilité du système pour toutes les valeurs possibles des coefficients. Cela revient à dire que nous avons la stabilité robuste si :

$$\begin{array}{l}\forall\alpha_1\in[\alpha_1], \forall\alpha_2\in[\alpha_2], \forall\alpha_3\in[\alpha_3], \forall\tau\in[\tau], \forall k\in[k],\\ a_3,\ a_2,\ \frac{a_2a_1-a_3a_0}{a_2} \text{ et } a_0 \text{ sont de même signe.}\end{array}$$

Or, pour quatre nombres u, v, w et x nous avons l'équivalence :

$$\begin{array}{rl} & u,\ v,\ w \text{ et } x \text{ sont de même signe}\\ \Leftrightarrow & \max\left(\min\left(u,v,w,x\right), -\max\left(u,v,w,x\right)\right)>0\end{array}$$

La condition de stabilité robuste revient donc à dire que la proposition :

$$\begin{array}{l}\exists\alpha_1\in[\alpha_1], \exists\alpha_2\in[\alpha_2], \exists\alpha_3\in[\alpha_3], \exists\tau\in[\tau], \exists k\in[k],\\ \max\left(\min\left(a_3,a_2,\frac{a_2a_1-a_3a_0}{a_2},a_0\right), -\max\left(a_3,a_2,\frac{a_2a_1-a_3a_0}{a_2},a_0\right)\right)\leq 0\end{array}$$

est fausse. Prouver la stabilité robuste de notre système revient donc à montrer que le CSP dont les variables, les domaines et les contraintes sont donnés respectivement par :

$$\begin{array}{rcl}\mathcal{V} & = & \{a_0,a_1,a_2,a_3,\alpha_1,\alpha_2,\alpha_3,\tau,k\},\\ \mathcal{D} & = & \{[\alpha_0],[\alpha_1],[\alpha_2],[\alpha_3],[\alpha_2],[\alpha_3],[\tau],[k]\},\\ \mathcal{C} & = & \left\{\begin{array}{l} a_3=\tau+\alpha_3k\ ;\ a_2=\alpha_2k+2\alpha_3+1\ ;\ a_1=\alpha_3-\alpha_1\tau+2\alpha_2,\\ a_0=-\alpha_1+\alpha_2\ ;\ b=\frac{a_2a_1-a_3a_0}{a_2};\\ \max\left(\min\left(a_3,a_2,b,a_0\right), -\max\left(a_3,a_2,b,a_0\right)\right)\leq 0.\end{array}\right\}\end{array}$$

possède un ensemble solution vide.

Supposons que les valeurs pour les coefficients de la moto, du régulateur et du capteur satisfont :

$$\alpha_1\in[8,8;9,2], \alpha_2\in[2,8;3,2], \alpha_3\in[0,8;1,2], \tau\in[1,8;2,2], k\in[-3,2;-2,8],$$

alors IntervalPeeler comme RealPaver prouve la stabilité robuste du système bouclé. Pour le programme source donné ci-dessous, nous obtenons un ensemble vide.en $10^{-3}s$

```
Input Variables
   a0      in  ]-oo,+oo[
   a1      in  ]-oo,+oo[
   a2      in  ]-oo,+oo[
   a3      in  ]-oo,+oo[
   alpha1  in  [8.8,9.2]
   alpha2  in  [2.8,3.2]
   alpha3  in  [0.8,1.2]
   tau     in  [1.8,2.2]
   k       in  [-3.2,-2.8]
   b       in  ] -oo,+oo[
Output Variables
Constraints
   a3 = tau + alpha3*k
   a2 = alpha2*k + 2*alpha3 + 1
   a1 = alpha3 - alpha1*tau + 2*alpha2
   a0 = alpha2 - alpha1
   b = (a2*a1 - a3*a0)/a2
   0 >= max( min(a3,a2,b,a0)))
         -max(a3,a2,b,a0))))
End
```

5.4 Estimation d'état

Prenons un système dans sa représentation d'état. Son équation d'évolution est :

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \mathbf{w}) \tag{5.16}$$

où $\mathbf{x}$ est l'état du système et $\mathbf{u}$ l'entrée du système. Son équation d'observation :

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{w}) \tag{5.17}$$

modélise les sorties mesurables $\mathbf{y}$ du système. Le modèle du système et de l'observation n'étant pas parfait et les mesures ayant une possibilité d'erreur due aux capteurs, il est possible d'introduire une erreur ou perturbation $\mathbf{w}$ dans le modèle qui permet non pas de modéliser mais de

prendre en compte tous ces facteurs d'erreur.

Pour réaliser l'estimation d'état, le modèle d'état étant supposé connu, nous mesurons les valeurs de $\mathbf{y}$ et cherchons à retrouver l'état du système $\mathbf{x}$.

Pour ce faire le système dynamique est discretisé à l'ordre 1 et devient :

$$\mathbf{x}(k+1) = \mathbf{f}((\mathbf{x}(k), \mathbf{u}(k), \mathbf{w}(k)), \tag{5.18}$$

$$\mathbf{y}(k) = \mathbf{g}(\mathbf{x}(k), \mathbf{w}(k)), \tag{5.19}$$

où l'état $\mathbf{x}(k)$, les mesures $\mathbf{y}(k)$, l'entrée $\mathbf{u}(k)$ et les perturbations $\mathbf{w}(k)$ sont donnés à l'instant k. Le temps discret k variant de l'instant initial 0 de début des mesures jusqu'à l'instant de fin de mesure k_{max}. Lors de cette étape de discrétisation, une erreur s'ajoute pour l'équation d'évolution (une erreur dite de discrétisation), nous la considérons ici comme faisant partie de $\mathbf{w}(k)$.

Pour illustrer l'estimation d'état, prenons l'exemple du satellite [36] et définissons le CSP associé à ce problème.

Le satellite se déplace autour de la Terre sur laquelle est fixée un repère Ox_1x_2. L'état $\mathbf{x}$ est donc composé du déplacement x_1 suivant l'axe Ox_1, du déplacement x_2 suivant l'axe Ox_2 et des vitesses de déplacement x_3 x_4 suivant Ox_1et Ox_2. L'équation d'évolution est donc la suivante :

$$\begin{aligned} \dot{x_1} &= x_3, \\ \dot{x_2} &= x_4, \\ \dot{x_3} &= -\frac{x_2}{\sqrt{x_1^2 + x_2^2}^3}, \\ \dot{x_4} &= -\frac{x_1}{\sqrt{x_1^2 + x_2^2}^3}. \end{aligned} \tag{5.20}$$

Les mesures possibles pour localiser un satellite depuis la Terre sont les suivantes : y_1 l'angle entre la direction du satellite et une direction fixe, y_2 la vitesse radiale du satellite par effet Doppler et y_3 la distance Terre satellite au carré. A partir de la position du satellite et de sa vitesse, les relations permettant d'obtenir les mesures sont les suivantes :

$$\begin{aligned} y_1 &= \frac{x_2}{x_1}, \\ y_2 &= x_1x_3 - x_2x_4, \\ y_3 &= x_1^2 + x_2^2. \end{aligned} \tag{5.21}$$

Les mesures y_m sont effectuées sur le satellite et permettent d'obtenir les domaines d'appartenance pour les variables y :

$$[y] = [y_m - 0,1; y_m + 0,1] \tag{5.22}$$

De la même manière, l'erreur associée à l'équation d'évolution $w(k)$ est prise comme appartenant à l'intervalle $[-0,01; 0,01]$. A chacune des composantes i de l'état et pour chaque point de mesure

j est ajoutée la variable d'erreur ex_ip_j correspondant à cette erreur $w(k)$.

Les domaines de recherche pour les variables d'état du système sont prises entre $]-\infty;+\infty[$ sauf pour l'état initial où deux positions symétriques existent sur $\mathbb{R}$ du fait des mesures réalisées sur le satellite. Les domaines sont donc restreints grâce à la connaissance à priori sur le point de départ.

Le CSP associé est donc :

```
Input Variables
   x1p0 in [0,+oo]
   x2p0 in [0,+oo]
   x3p0 in [-oo,0]
   x4p0 in [0,+oo]
   ex1p0 in [-0.01,0.01]
   ex2p0 in [-0.01,0.01]
   ex3p0 in [-0.01,0.01]
   ex4p0 in [-0.01,0.01]
   ex1p1 in [-0.01,0.01]
   .../...
   ex4p200 in [-0.01,0.01]
Output Variables
   y1p0  in [-0.03156,0.16844]
   y2p0  in [-0.642605,-0.442605]
   y3p0  in [16.0892,16.2892]
   x1p1 in [-oo,+oo]
   x2p1 in [-oo,+oo]
   x3p1 in [-oo,+oo]
   x4p1 in [-oo,+oo]
   .../...
   x4p201 in [-oo,+oo]
Constraints
   x1p1=x1p0+0.5*x3p0+ ex1p0
   x2p1=x2p0+0.5*x4p0+ ex2p0
   x3p1=x3p0-0.5*x1p0/(sqrt(x1p0^2+x2p0^2))^3+ex3p0
   x4p1=x4p0-0.5*x2p0/(sqrt(x1p0^2+x2p0^2))^3+ex4p0
   y1p0=x2p0/x1p0
   y2p0=x1p0*x3p0+x2p0*x4p0
   y3p0=x1p0^2+x2p0^2
   .../...
   y3p200=x1p200^2+x2p200^2
```

```
End
```

La résolution de problème par IntervalPeeler est visualisé sur la figure 5-6. Les deux positions x_1 et x_2 sont représentées avec la trajectoire simulée du satellite sur la figure de gauche et les pavés $[x_1] \times [x_2]$ déterminés pour chaque instant k à partir des mesures $[\mathbf{y}(k)]$. Pour certains instants k (aux alentours de $k = 100$ et $k = 200$), la boite obtenue pour $[\mathbf{x}(k)]$ est peu précise. Pour renforcer cette précision, deux méthodes sont possibles : soit trouver des méthodes de consistance plus forte afin de s'approcher de la consistance globale pour ce problème (pour cette notion voir page 47) soit l'ajout dans le problème de mesures supplémentaires. En effet, d'autres mesures sont possibles sur le satellite, ces mesures permettant de lever le pavé d'équilibre de la propagation en supprimant de nouveaux domaines non consistants.

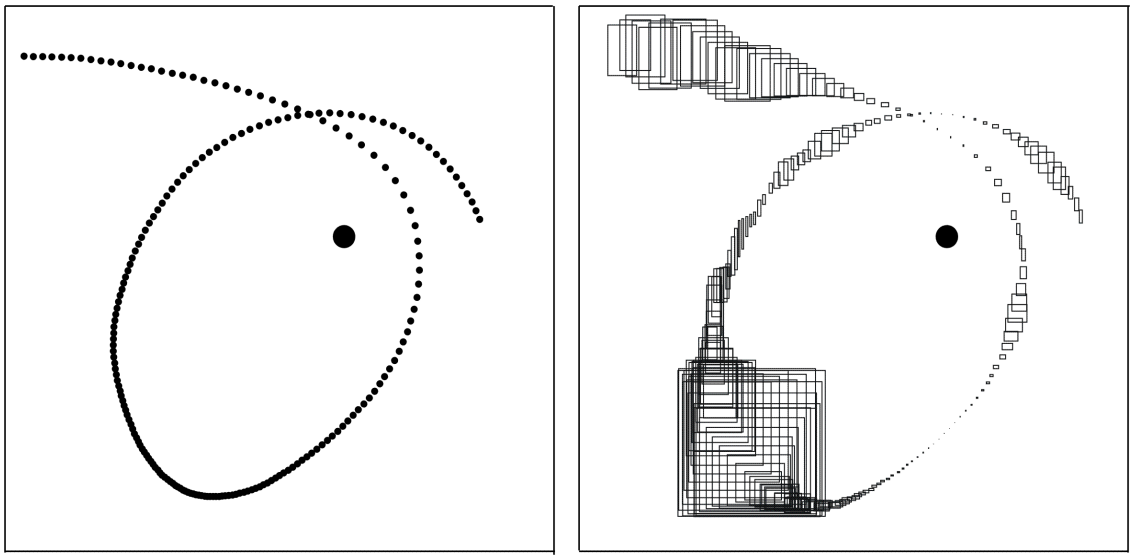

Fig. 5-6 – Illustration du parcours du satellite, la terre étant signalée par un point, la trajectoire simulée est représentée à gauche, celle déterminée par IntervalPeeler à droite.

Le comportement d'IntervalPeeler pour la résolution de ce CSP peut être comparé à celui du logiciel RealPaver. Les temps de calcul présentés dans le tableau 5.1 sont du même ordre de grandeur avec un avantage pour RealPaver qui ne nécessite pas un temps de précompilation des contraintes pour les formaliser en DAG.

	IntervalPeeler	RealPaver
$k_{max} = 200$	1.6e3ms	1e3ms
$k_{max} = 800$	7.26e3ms	12e3ms

Tableau 5.1 - Comparaison des temps de calcul pour différentes valeurs de k_{max}.

Le temps obtenu pour IntervalPeeler est aussi lié à la façon dont l'utilisateur choisit les va-

riables entrées-sorties du problème pour la formalisation du DAG. En effet, traitons le problème d'estimation jusqu'à l'instant $k_{max} = 200$. Le formalisme choisissant uniquement l'état à l'instant 0 et l'ensemble des vecteurs d'erreur comme variables d'entrées, le reste des variables étant des variables de sorties nécessite 35 propagations rétro-propagations et $1.6e3ms$ pour obtenir le pavé d'équilibre. Maintenant si toutes les variables sont choisies comme variables d'entrées, 243 propagations rétro-propagations sont nécessaires et le temps de résolution $17.77e3ms$ se voit multiplié d'un facteur 10 pour IntervalPeeler . Le choix des variables d'entrées-sorties est donc primordial pour obtenir l'algorithme le plus rapide.

5.5 Détection de défauts

Le problème de détection de défauts [8, 18] se formalise comme le problème de l'estimation d'état. Le système possède une équation d'évolution, l'état $\mathbf{x}$ du système n'est pas connu mais n'est pas recherché en soi. Dans le problème de détection de défauts, c'est l'instant où l'état prend des valeurs aberrantes que nous cherchons à estimer. La propagation de contraintes signalant le défaut lorsqu'un domaine intervalle associé à une variable prend pour valeur l'ensemble vide.

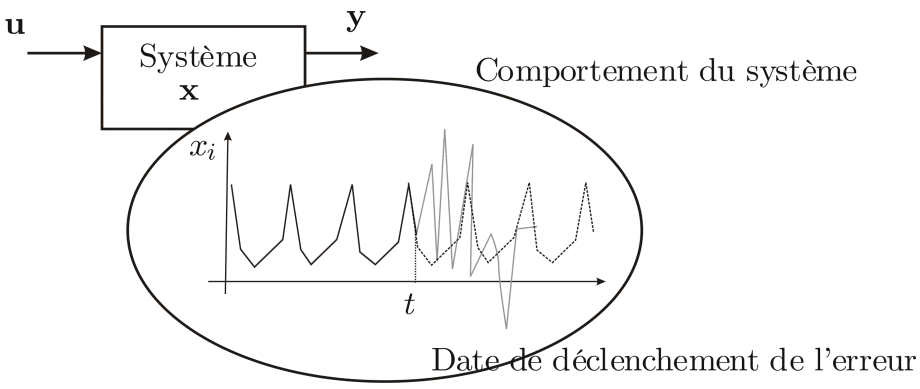

Fig. 5-7 – Représentation d'un système présentant un défaut à l'instant T.

Sur un système S observé, la présence d'un défaut va coïncider avec un changement du comportement du système. Considérons un système qui, pendant son fonctionnement normal, suit l'équation d'évolution discrétisée suivante :

$$x_1(k+1) = \sin(x_2(k)) + \cos(x_2(k)) + ex_1(k), \quad (5.23)$$

$$x_2(k+1) = \frac{x_2(k) - x_3(k)}{x_2(k)} + ex_2(k), \quad (5.24)$$

$$x_3(k+1) = 1 + x_3(k) + (x_3(k))^2 - (x_3(k)^3) + ex_3(k). \quad (5.25)$$

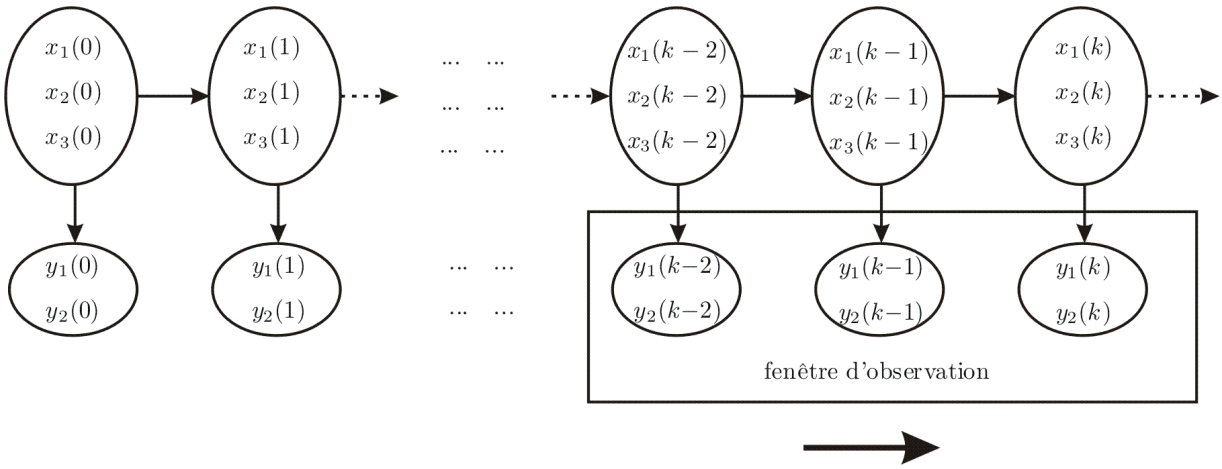

FIG. 5-8 – Illustration du principe de fenêtre glissante d'horizon trois dans cet exemple.

Le système étant observé grâce aux sorties y_1 et y_2 données par l'équation d'observation :

$$y_1(k) = \sin(x_1(k)) + \sin(x_2(k)) + \sin(x_3(k)), \tag{5.26}$$

$$y_2(k) = x_1(k)) + x_2(k). \tag{5.27}$$

A l'instant où débute le défaut, le système change alors d'équation d'évolution, dans l'exemple 1 l'équation (5.24) devient :

$$x_2(k+1) = \tan(\frac{x_2(k) + x_3(k)}{x_2(k)}) + ex_2(k), \tag{5.28}$$

et dans le deuxième exemple, l'erreur est plus faible, elle correspond juste au changement de la valeur d'un paramètre de l'équation (5.25) :

$$x_3(k+1) = 1.1 + x_3(k) + (x_3(k))^2 - (x_3(k)^3) + ex_3(k).$$

Le problème est donc de détecter l'erreur au plus tôt, c'est-à-dire détecter quand l'état estimé avec le modèle normal sera aberrant par rapport aux mesures effectuées sur le système réel. Pour ce problème de détection de défaut, il est possible de réaliser une fenêtre glissante avec quelques instants discrets pour horizon (voir figure 5-8). Un CSP statique est défini par les équations d'évolution et d'observation sur, par exemple, trois unités de temps, et par les variables prenant pour domaines $[\mathbf{x}(0)]$ pour l'état initial, $[\mathbf{y}(0)]$,$[\mathbf{y}(1)]$...$[\mathbf{y}(3)]$ pour les mesures et $[-0,01; 0,01]$ pour les erreurs. Les états aux instants suivant $k = 1, 2, 3$ n'étant pas connus, les domaines sont pris à $]-\infty; +\infty[$. Ce CSP correspond au fichier source suivant :

```
Input variables
   x1pt0 in [-9.99,10.01]
```

```
   x2pt0 in [-9.99,10.01]
   x3pt0 in [-10.01,9.99]
   ex1pt1 in [-0.01,0.01]//erreur de modele
   ex2pt1 in [-0.01,0.01]
   ex3pt1 in [-0.01,0.01]
   ex1pt2 in [-0.01,0.01]
   ../..
   ex3pt3 in [-0.01,0.01]
Output variables
   y1pt0 in [-1.6666583333574e-07,0.019999833334167]
   y2pt0 in [0.01,0.03]
   x1pt1 in [-oo,+oo]
   x2pt1 in [-oo,+oo]
   x3pt1 in [-oo,+oo]
   y1pt1 in [2.5821839809328,2.6021839809328]
   ../..
   y2pt3 in [-1.611443772012,-1.591443772012]
Constraints
   y1pt0=sin(x1pt0)+sin(x2pt0)+sin(x3pt0)//mesure sortie n=0
   y2pt0=x1pt0+x2pt0
   x1pt1=sin(x2pt0)+cos(x2pt0)+ex1pt1//calcul de l etat n=1
   x2pt1=((x2pt0-x3pt0)/(x2pt0))+ex2pt1
   x3pt1=1+x3pt0+(x3pt0)^2-(x3pt0)^3+ex3pt1
   y1pt1  = sin(x1pt1)+sin(x2pt1)+sin(x3pt1)//mesure en sortie n=1
   y2pt1  = x1pt1+x2pt1
   ../..
   y2pt3  = x1pt3+x2pt3
End
```

La résolution du problème de détection de défaut est alors prise en compte en traitant un CSP différent à chaque instant. Lorsque l'instant $k = 3$ est passé, l'ensemble des variables et des domaines correspondant à l'instant $k = 0$ sort du CSP pour laisser entrer les variables et domaines au temps $k = 4$. Le CSP est alors résolu par propagation de contraintes, s'il est toujours consistant, la fenêtre d'observation se déplace vers l'instant suivant. Ce processus se poursuit tant que l'ensemble vide, signe de la présence d'un défaut, n'est pas obtenu.

Sur les deux essais que nous avons effectués, la détection du défaut a été observée. Dans le premier exemple où l'équation d'évolution subie un changement brutal (cf équation 5.28), l'ensemble vide est obtenu dès le déplacement de la fenêtre d'observation sur l'instant de déclenchement du défaut. Dans l'autre exemple, où seul un changement de la valeur d'un paramètre intervient dans une équation d'évolution, la détection n'est pas immédiate. Il faut attendre l'instant $k + 5$

pour détecter le défaut et obtenir l'ensemble vide.

La propagation de contraintes permet donc sur ces exemples de détecter la présence d'un défaut. Une étude reste à faire pour savoir si les instants de détection du défaut obtenus sont les instants de détection au plus tôt. En effet, l'ensemble de solutions défini par le CSP sur la fenêtre glissante peut être vide alors que la propagation de contraintes est bloquée sur un pavé d'équilibre dû à une consistance locale.

5.6 Résultats pour l'étalonnage

Le problème de l'étalonnage présenté chapitre 2 est un problème d'estimation de paramètres où nous recherchons les valeurs des paramètres $\mathbf{p}$ connaissant un jeu mesure de 50 configurations du robot. Pour chacune des configurations, les valeurs pour les variables articulaires positionnant le robot sont connues ainsi que les mesures $\mathbf{x}$ sur trois points fixés sur l'organe terminal.

La connaissance du modèle géométrique direct permet alors de générer le CSP et pour IntervalPeeler le fichier source suivant :

```
Input variables
   t0 in [1.5708,1.57081]//parametres
   r0 in [0.4,0.6]
   a1 in [0.1,0.10001]
   ../..
   bz3 in [0,0.2]
   P1q1 in [-3.11668,-3.11666]  //variables articulaires
   P1q2 in [-1.03576,-1.03574]
   P1q3 in [-2.93258,-2.93256]
   P1q4 in [-0.906529,-0.906509]
   P1q5 in [-1.77689,-1.77687]
   P1q6 in [0.232291,0.232311]
   ../..
   P50q6 in [-3.12741,-3.12739]
Output variables
   P1ax1 in [0.231962,0.232162]  //mesures outil
   P1ay1 in [-0.568294,-0.568094]
   P1az1 in [0.470933,0.471133]
   P1ax2 in [0.265051,0.265251]
   P1ay2 in [-0.663951,-0.663751]
   P1az2 in [0.569744,0.569944]
   P1ax3 in [0.206537,0.206737]
```

```
   P1ay3 in [-0.705417,-0.705217]
   P1az3 in [0.44797,0.44817]
   ../..
   P50az3 in [0.247923,0.248123]
Constraints
   l1=sqr(bx1-bx2)+sqr(by1-by2)+sqr(bz1-bz2)
   l2=sqr(bx2-bx3)+sqr(by2-by3)+sqr(bz2-bz3)
   l3=sqr(bx3-bx1)+sqr(by3-by1)+sqr(bz3-bz1)
   x0=bx1*cos(P1q6)+by1*(-sin(P1q6))
   y0=bx1*sin(P1q6)+by1*cos(P1q6)
   z0=bz1  bx1=x0*cos(P1q6)+y0*sin(P1q6)
   bx1=x0*cos(P1q6)+y0*sin(P1q6)
   by1=x0*(-sin(P1q6))+y0*cos(P1q6)
   tan(P1q6)=(y0*bx1-x0*by1)/(x0*bx1+y0*by1)
   sqr(bx1)+sqr(by1)=sqr(x0)+sqr(y0)
   ../..
   tan(t0)=(y3749*x3748-x3749*y3748)/(x3749*x3748+y3749*y3748)
   sqr(x3748)+sqr(y3748)=sqr(x3749)+sqr(y3749)
End
```

Pour traiter le problème de l'étalonnage du robot STAUBLI rx90 avec les intervalles, l'idéal serait de trouver la plus petite boîte extérieure encadrant l'ensemble de solutions pour chacun des paramètres de construction du robot. Dans la réalité, trouver cette boîte est difficile et le résultat est souvent une boîte que nous ne savons pas situer proche ou non de la boîte minimale.

Pour obtenir les premiers résultats, une propagation de contraintes classique est effectuée sur 4 CSP correspondant à l'étalonnage du robot STAUBLI RX. Les premiers tests, réalisés pour l'étalonnage de *tous les paramètres* $\mathbf{p}$, ne permettent pas d'obtenir une contraction des domaines de recherche. La formulation ne permet pas non plus de lever le pavé fixe initial.

Les deux tests suivants, avec le CSP de la formulation matricielle et le CSP pour la contrainte angle, sont générés en suivant l'ensemble de la procédure de simulation présentée dans le chapitre 2 où l'étalonnage est limité aux longueurs mal connues et aux longueurs non nulles des paramètres du robot. Tous les autres paramètres n'étant pas étalonnés sont supposés pour cet essai connus avec une précision de 10^{-5}. L'arrêt de la propagation s'effectue quand la contraction du vecteur d'intervalles représentant les variables n'est plus que d'un facteur 10^{-5} et permet d'obtenir les domaines suivants :

	domaines de recherche	domaines obtenus avec la formulation matricielle	domaines obtenus avec la contrainte angle	$\frac{\text{taille2}}{\text{taille1}}$
r_0	$[0,4;0,6]$	$[0,488772;0,501644]$	$[0,494046;0,50101]$	0, 54
d_1	$[0;0,1]$	$[0;0,000964102]$	$[0;0,000558009]$	0, 58
r_1	$[0;0,1]$	$[0;0,00911161]$	$[0;0,00693694]$	0, 76
d_3	$[0,49;0,51]$	$[0,497796;0,50185]$	$[0,498385;0,501133]$	0, 68
r_4	$[0,49;0,51]$	$[0,499031;0,502242]$	$[0,499216;0,50114]$	0, 60
b_x^1	$[0;0,2]$	$[0,0993651;0,10123]$	$[0,0996052;0,100629]$	0, 55
b_y^1	$[0,1;0,3]$	$[0,199615;0,200586]$	$[0,199502;0,200455]$	0, 98
b_z^1	$[0;0,2]$	$[0,0992392;0,100921]$	$[0,0997107;0,100714]$	0, 60
b_x^2	$[0;0,2]$	$[0,0993244;0,101355]$	$[0,0996747;0,100712]$	0, 51
b_y^2	$[0;0,2]$	$[0,099531;0,100757]$	$[0,0994585;0,10031]$	0, 69
b_z^2	$[0,1;0,3]$	$[0,199172;0,200965]$	$[0,199535;0,200642]$	0, 62
b_x^3	$[0,1;0,3]$	$[0,199372;0,201203]$	$[0,199689;0,200578]$	0, 49
b_y^3	$[0;0,2]$	$[0,0995955;0,100753]$	$[0,0997562;0,100319]$	0, 49
b_z^3	$[0;0,2]$	$[0,0991525;0,100918]$	$[0,0995661;0,100557]$	0, 56

Tableau 5.2 - Comparatif donnant les résultats de l'étalonnage pour les deux formalisations.

Le tableau 5.2 nous donne les résultats pour l'étalonnage des 14 paramètres de longueurs du robot Staubli Rx90. Il présente pour chaque variable l'intervalle de recherche, l'intervalle obtenu par simple propagation et celui obtenu en utilisant la contrainte angle. La contraction des domaines de recherche est un chiffre peu significatif, en effet la taille du domaine de recherche importe peu pour cette méthode de calcul. En revanche la taille des intervalles obtenus, autrement dit l'incertitude calculée autour de nos paramètres, est intéressante. Cette taille varie $0,01m$ pour les paramètres les moins contractés à $0,001m$. Une incertitude d'$1cm$ reste inacceptable pour un problème d'étalonnage, celle d'$1mm$ un résultat de médiocre qualité.

Le deuxième formalisme intégrant la contrainte angle permet une amélioration de la contraction de plus de 30%. La dernière colonne du tableau présente ce rapport entre les tailles des intervalles obtenus avec les deux formalismes. Le temps de résolution de $137,3s$ est supérieur à celui du premier formalisme qui trouve le pavé d'équilibre en $8,34s$. Ce temps plus long peut s'expliquer par le rajout de nombreuses contraintes redondantes. Il augmente le nombre de contraintes primitives et oblige la procédure de propagation de contraintes à balayer l'ensemble des contraintes primitives plusieurs fois de plus. La mise en place de la contrainte globale angle, traitée non plus avec des contraintes redondantes mais directement, permettra d'obtenir si ce n'est un meilleur résultat, du moins le même qu'avec les contraintes redondantes dans un temps beaucoup plus court.

D'autres tests sur des CSP avec des domaines de recherche pour un étalonnage d'uniquement 4 ou 7 angles permettent une contraction de ces angles. Le nombre d'angles contractés augmente lors de l'utilisation du deuxième formalisme. Cette progression est encourageante pour la mise en place de l'étalonnage sur l'ensemble des paramètres mais reste encore loin des 30 paramètres

de l'étalonnage total.

Hormis la possible amélioration grâce à la contrainte angle, la mise en place d'une technique de résolution qui prend en compte les contraintes de façon globale pour un ensemble de contraintes est une piste qui peut permettre de lever le blocage sur un pavé fixe lors de la propagation de contraintes. L'autre piste étant l'utilisation de méthodes de consistance plus forte, c'est-à-dire celle que nous abordons dans le chapitre suivant.

Chapitre 6

Ajout de contraintes pour renforcer la propagation

La réalisation du DAG a permis l'obtention rapide du pavé d'équilibre de la méthode de propagation de contraintes. Ce pavé d'équilibre permet d'obtenir une information sur l'ensemble de solutions, mais en général il ne correspond pas à la boîte la plus petite encadrant l'ensemble de solutions. Le but est donc de trouver des méthodes permettant d'obtenir si possible la boîte optimale ou du moins une boîte aussi petite que possible.

Dans ce chapitre nous allons donc illustrer le problème de consistance locale due à la multi-occurrence des variables dans les contraintes. Une méthode pour lutter contre ce phénomène est la box consistance ([70],[10]...). Elle peut être vue comme une méthode d'ajout dynamique de contraintes permettant une résolution du problème.

6.1 Le problème de la consistance locale

Prenons le CSP suivant :

$$\mathcal{H}_0 : \left\{\mathcal{V}_0 = \{x\}, \mathcal{D}_0 = \{[x] =]-\infty; +\infty[\}, \mathcal{C}_0 = \{x^3 + x = 0\}\right\}. \tag{6.1}$$

Ce CSP $\mathcal{H}_0$ n'est pas celui qui va être traité par la méthode de propagation de contraintes. En effet la contrainte $x^3 + x = 0$ n'est pas primitive. Nous pouvons donc écrire le CSP équivalent $\mathcal{H}_1$ qui correspond à la décomposition en contraintes primitives de $\mathcal{H}_0$:

$$\mathcal{H}_1 : \begin{cases} \mathcal{V}_1 &= \{x, y\}, \\ \mathcal{D}_1 &= \{[x] =]-\infty; +\infty[, [y] =]-\infty; \infty[\}, \\ \mathcal{C}_1 &= \{y = x^3, \quad y + x = 0\}. \end{cases} \tag{6.2}$$

La propagation de contraintes sur ces deux contraintes possède trois pavés d'équilibre : $]-\infty; +\infty[\times]-\infty; +\infty[$, $[-1; 1] \times [-1; 1]$, $[0] \times [0]$. En effet quand $x \in [-1; 1]$ nous obtenons

$y \in [-1;1]$ et aucune des deux contraintes ne peut lever l'ambiguïté. L'unique solution du CSP avec $]-\infty;+\infty[$ pour domaine pour x, étant 0, la boîte obtenue n'est pas consistante pour les contraintes prises globalement mais consistante localement vis-à-vis de chacune des contraintes primitives.

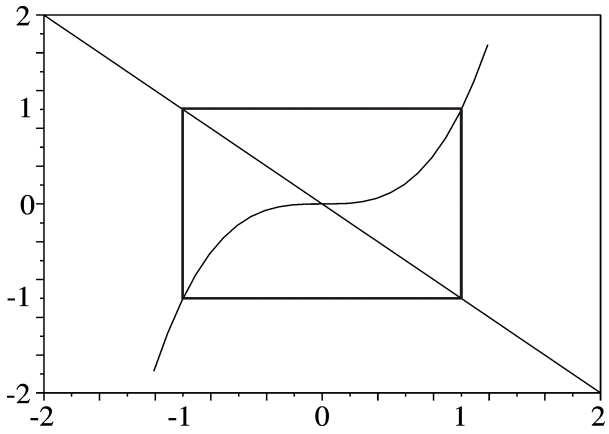

FIG. 6-1 – Illustration du problème de consistance locale

Remarque 6.1.1 *Dans cet exemple, le problème de consistance locale peut aussi se voir comme un problème de multi-occurrence des variables. Dans $\mathcal{H}_0$, x apparaît deux fois dans la contrainte $x^3+x=0$, et empêche l'obtention de la boite optimale (présentée au chapitre 3). Les algorithmes proposés dans ce chapitre vont principalement lutter contre la multi-occurrence des variables d'une contrainte, une des facettes du problème de consistance locale.*

6.2 Ajout de contraintes statiques

Un des moyens pour lever ce pavé d'équilibre dû à une consistance locale est d'ajouter au CSP des contraintes redondantes [68]. Dans l'exemple précédent une manipulation formelle permet de générer de nouvelles contraintes par exemple, en factorisant x, nous obtenons la contrainte :

$$x.(\frac{y}{x}+1)=0 \tag{6.3}$$

en remplaçant y dans l'expression, nous obtenons alors la contrainte redondante :

$$x.(1+x^2)=0 \tag{6.4}$$

Cette contrainte va permettre de lever les deux points fixes précédents. Si $x \in]-\infty;+\infty[$, alors $1+x^2 \in [1;+\infty[$. La contrainte primitive correspondant au produit de y avec cette expression permet alors d'obtenir l'intervalle [0] pour y. De même, quand $x \in [-1;1]$, $1+x^2 \in [1;2]$ or $y.(1+x^2) \in [0]$, d'où $y \in [0]$.

Dans cet exemple la génération de contraintes statiques reste assez aisée, mais pour un CSP de grande dimension, il est difficile de générer des bonnes contraintes redondantes. En effet, parmi l'infinité de contraintes que nous pouvons générer grâce au calcul formel, comment choisir sans savoir à l'avance si elles permettront une meilleure contraction.

Une possibilité reste la génération automatique de contraintes réduisant le nombre d'occurrences des variables. L'autre possibilité est l'utilisation de contraintes dynamiques permettant de s'affranchir de la multi-occurrence d'une variable. C'est ce deuxième choix qui est présenté dans la suite.

6.3 Ajout de contraintes dynamiques

Soit le CSP suivant :

$$\mathcal{H}_0 : \{\mathcal{V}_0 = \{x_1, x_2, \ldots, x_n\}, \mathcal{D}_0 = \{[x_1], [x_2], \ldots, [x_n]\}, \mathcal{C}_0 = \{f(x_1, x_2, \ldots, x_n) = 0, \ldots\}\}. \quad (6.5)$$

En appliquant le théorème de la valeur moyenne, nous pouvons déduire de la première contrainte les $2n$ contraintes suivantes :

$$\begin{array}{rll} f(x_1^-, x_2, x_3, \ldots) + \frac{\partial f}{\partial x_1}(a_1, x_2, x_3, \ldots).(x_1 - x_1^-) & = 0, & a_1 \in [x_1], \\ f(x_1^+, x_2, x_3, \ldots) + \frac{\partial f}{\partial x_1}(a_2, x_2, x_3, \ldots).(x_1 - x_1^+) & = 0, & a_2 \in [x_1], \\ f(x_1, x_2^-, x_3, \ldots) + \frac{\partial f}{\partial x_2}(x_1, a_2, x_3, \ldots).(x_2 - x_2^-) & = 0 & a_3 \in [x_2], \\ \ldots & & \end{array} \quad (6.6)$$

Remarque 6.3.1 *D'autres contraintes comme :*

$$\begin{array}{rll} f(x_1^-, x_2, \ldots) + \frac{\partial f}{\partial x_1}(x_1^-, x_2, \ldots).(x_1 - x_1^-) & & \\ \quad + \frac{1}{2}\frac{\partial^2 f}{\partial x_1^2}(b_1, x_2, \ldots).(x_1 - x_1^-)^2 & = 0, & b_1 \in [x_1], \end{array} \quad (6.7)$$

ou :

$$\begin{array}{rll} f(x_1^-, x_2^-, x_3^-, \ldots) + \frac{\partial f}{\partial x_1}(c_1, x_2^-, x_3^-, \ldots).(x_1 - x_1^-) & & c_1 \in [x_1], \\ \frac{\partial f}{\partial x_2}(x_1, c_2, x_3^- \ldots).(x_2 - x_2^-) + \ldots & = 0, & c_2 \in [x_2], \ldots \end{array} \quad (6.8)$$

peuvent aussi être générées par dérivations successives. Dans la suite, nous utiliserons uniquement les contraintes données par (6.6) pour illustrer le principe de la méthode. L'utilisation de plusieurs dérivées correspond à un compromis entre temps de calcul des dérivées, nombre de termes intervenants dans la nouvelle contrainte et contraction que nous pouvons obtenir, ce qui ne change en rien le principe des contraintes dynamiques.

Dans le cas des contraintes statiques, la génération de contraintes redondantes n'est pas systématique. Ici, l'usage des dérivées permet à partir de l'expression formelle de f d'obtenir automatiquement des contraintes redondantes (6.6). Par contre, celles-ci dépendent des bornes x^-

et x^+, *i.e.*, qu'elles dépendent des domaines des variables. Ce genre de contraintes n'est traditionnellement pas accepté dans les CSP à cause du risque de perdre des points solutions lors de la propagation. En effet si $[x]$ est contracté lors d'une étape de la propagation, il se peut que les valeurs prises par les variables a_1, a_2, b_1, b_2 soient changées. Or les variables sont supposées avoir une valeur constante pendant la propagation sur le CSP.

6.3.1 CSP Dynamiques

Pour manipuler des contraintes dépendantes des domaines des variables, il est possible d'utiliser la notion de *CSP dynamique* qui est un CSP où chaque contrainte peut être ajoutée ou supprimée pendant la propagation [15]. Ces contraintes sont dites *dynamiques*. Insistons tout de même sur le fait que même si ces contraintes dynamiques sont ajoutées au CSP, elles ne peuvent être modifiées. Les contraintes dynamiques ne dépendent donc toujours pas des domaines des variables. C'est l'ajout et le retrait de ces contraintes qui leur confère cette propriété dynamique. La contrainte étant vérifiée par toutes les solutions du CSP, elle permet une contraction. Une fois cette contraction réalisée, la contrainte n'est plus utile car toujours vérifiée, elle est donc retirée et ne ralentit pas la propagation.

Exemple 6.3.1 *Soit la contrainte suivante* $x_1^2 x_2 + x_2^2 = 0$ *et les domaines pour* x_1 *et* x_2 *où* $[x_1] = [-2; 4]$ *and* $[x_2] = [-1; 2]$. *Les contraintes redondantes (6.6) sont données par :*

$$\begin{array}{rll} f(x_1^-, x_2) + \frac{\partial f}{\partial x_1}(a_1, x_2).(x_1 - x_1^-) & = 0, & a_1 \in [x_1], \\ f(x_1^+, x_2) + \frac{\partial f}{\partial x_1}(a_2, x_2).(x_1 - x_1^+) & = 0, & a_2 \in [x_1], \\ f(x_1, x_2^-) + \frac{\partial f}{\partial x_2}(x_1, a_3).(x_2 - x_2^-) & = 0, & a_3 \in [x_2], \\ f(x_1, x_2^+) + \frac{\partial f}{\partial x_2}(x_1, a_4).(x_2 - x_2^+) & = 0, & a_4 \in [x_2], \end{array} \tag{6.9}$$

où $f(x_1, x_2) = x_1^2 x_2 + x_2^2$. *Or nous avons :*

$$\frac{\partial f}{\partial x_1} = 2x_1 x_2, \quad \frac{\partial f}{\partial x_2} = x_1^2 + 2x_2, \tag{6.10}$$

d'où la modification des contraintes :

$$\begin{array}{rll} \left(x_1^-\right)^2 x_2 + x_2^2 + \left(2a_1 x_2\right).(x_1 - x_1^-) & = 0, & a_1 \in [x_1], \\ \left(x_1^+\right)^2 x_2 + x_2^2 + \left(2a_2 x_2\right).(x_1 - x_1^+) & = 0, & a_2 \in [x_1], \\ x_1^2 x_2^- + \left(x_2^-\right)^2 + \left(x_1^2 + 2a_3\right).(x_2 - x_2^-) & = 0, & a_3 \in [x_2], \\ x_1^2 x_2^+ + \left(x_2^+\right)^2 + \left(x_1^2 + 2a_4\right).(x_2 - x_2^+) & = 0, & a_4 \in [x_2], \end{array} \tag{6.11}$$

ce qui donne avec le calcul de bornes :

$$\begin{array}{rll} 4x_2 + x_2^2 + \left(2a_1 x_2\right).(x_1 + 2) & = 0, & a_1 \in [-2; 4], \\ 16x_2 + x_2^2 + 2a_2 x_2.(x_1 - 4) & = 0, & a_2 \in [-2; 4], \\ -x_1^2 + 1 + \left(x_1^2 + 2a_3\right).(x_2 + 1) & = 0, & a_3 \in [-1; 2], \\ 2x_1^2 + 4 + \left(x_1^2 + 2a_4\right).(x_2 - 2) & = 0, & a_4 \in [-1; 2]. \end{array} \tag{6.12}$$

Ces quatre contraintes peuvent être insérées dans le CSP pour permettre la contraction des domaines par la propagation. Une fois cette contraction sur les domaines pour x_1 et x_2 effectuée, ces contraintes peuvent être supprimées du CSP. De nouvelles contraintes peuvent alors être générées avec de nouvelles valeurs pour x_i^- et x_i^+ et des nouveaux domaines plus petits pour a_i. Elles sont de nouveau insérées dans le CSP pour permettre une meilleure contraction.

6.3.2 Obtention du pavé optimal

Dans le CSP, supposons qu'il existe une contrainte de la forme $f(x) = 0$ où f ne dépend que de la variable x. A partir de l'équation (6.6), la contrainte dynamique suivante peut être générée avec :

$$f(x^-) + \frac{df}{dx}(a_1)(x - x^-) = 0, \qquad a_1 \in [x]. \tag{6.13}$$

Une propagation sur cette contrainte peut se voir comme l'algorithme intervalle suivant :

Algorithme PROPAG-1D(entrée sortie : $[x]$)
1 $[z_1] = [x] - x^-$,
2 $[z_2] = [f']([x])$,
3 $[z_3] = f(x^-) + [z_2] * [z_1]$,
4 si $0 \notin [z_3]$ alors $[x] = \emptyset$,
5 sinon $[z_1] = -(f(x^-)/[z_2]) \cap [z_1]$,
6 $[x] := ([z_1] + x^-) \cap [x]$.

A l'étape 2, $[f']([x])$ représente la fonction d'inclusion naturelle [58] de $\frac{df}{dx}$. Selon les domaines $[x]$, à l'étape 5 la division peut générer deux intervalles disjoints. Dans ce cas, ces intervalles sont séparés par zéro. Or comme $[z_1]$ est positif, l'intersection donnera toujours un intervalle positif. Nous aurons donc dans tous les cas une contraction de $[z_1]$ à l'étape 5 sauf si $f(x^-) = 0$ ou si $[z_2]$ est non borné. Par conséquent, si f est une fonction différentiable (*ie.*, $[z_2]$ est fini) et si $f(x^-) \neq 0$, $[x]$ sera toujours contracté.

Une illustration graphique de cette procédure de contraction est donnée sur la Figure (6-2) avec pour contrainte $\cos(x)+\sin(2x) = 0$ et pour domaine pour x, l'intervalle $[0; 5]$. Ce domaine initial pour $[x]$ est alors contracté tant que sa borne inférieure x^- vérifie $f(x^-) \neq 0$. A chaque étape, l'évaluation de f au point x^- et la fonction d'inclusion naturelle pour $\frac{df}{dx}$ permet de représenter un cône dans lequel la fonction varie. $[f]([x])$ appartient donc à ce cône. Mais la contrainte est $f(x) = 0$, ce qui permet de réduire le domaine pour $[x]$ à l'intersection avec le cône. Les points à l'extérieur ne vérifiant pas $f(x) = 0$.

Ce résultat peut se formuler sous le théorème suivant :

Théorème 6.1 Une contrainte de la forme $f(x) = 0$, où $f(x)$ est différentiable sur $[x]$ si le domaine $[x]$ de la variable x est borné, alors une propagation avec les deux contraintes dyna-

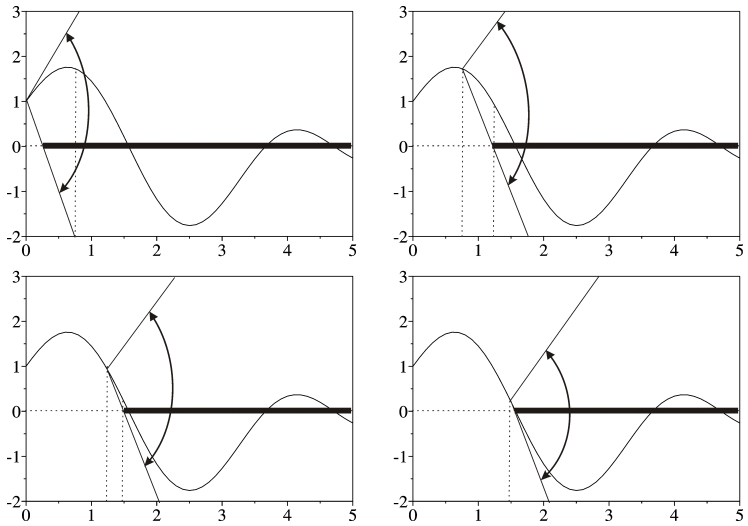

FIG. 6-2 – Illustration de la procédure de contraction associée à la contrainte dynamique (6.13).

miques :

$$\begin{array}{ll} f(x^-) + \frac{df}{dx}(a_1)(x - x^-) & = 0 \qquad a_1 \in [x], \\ f(x^+) + \frac{df}{dx}(a_2)(x - x^+) & = 0 \qquad a_2 \in [x], \end{array} \tag{6.14}$$

converge vers le plus petit intervalle contenant toutes les solutions de l'équation $f(x) = 0, x \in [x]$.

L'algorithme obtenu est très proche de l'algorithme en 1-dimension de newton par intervalle proposé par [11] qui permet d'obtenir le plus grand pavé d'un CSP box-consistant.

Exemple 6.3.2 *Prenons la contrainte :*

$$f(x) = 5\cos(x) + \sin(2x) + \sin(3x) + \sin(4x^2), \tag{6.15}$$

où $x \in [x] = [2; 7]$. *Le domaine obtenu après une propagation classique est* $x \in [2; 5{,}36]$. *Ce n'est manifestement pas le plus petit intervalle contenant toutes les solutions de* $f(x) = 0$. *En effet,* x *apparaît quatre fois dans l'expression de* f *et ces multi-occurrences bloquent la propagation. En ajoutant deux contraintes dynamiques données par l'équation (6.14), la contraction converge vers le plus petit intervalle* $[4, 3596; 4, 6644]$ *(voir Figure 6-3), comme prévu par le théorème 6.1.*

En général pour nos applications, nous avons rarement des contraintes monovariables. Néanmoins ce théorème illustre la capacité des contraintes dynamiques générées à partir de la dérivée, à palier efficacement au problème de multi-occurrence des variables dans les contraintes.

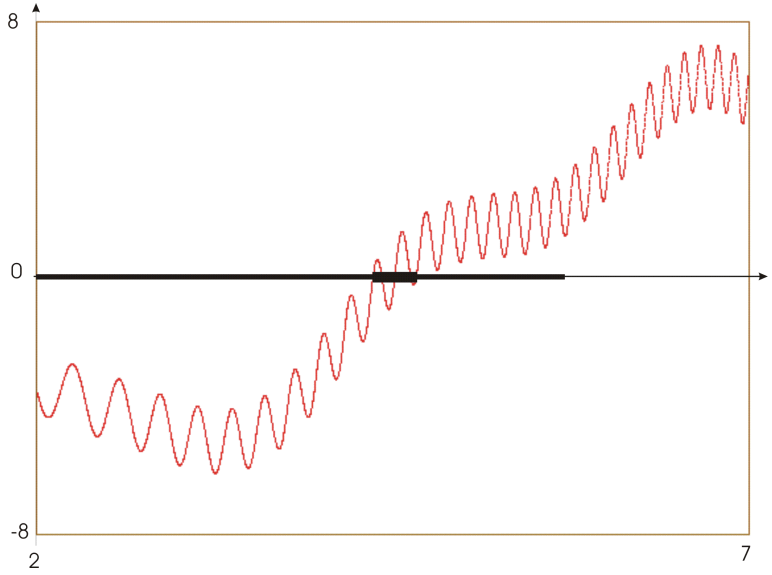

FIG. *6-3 – Contraction avec la propagation de contrainte classique (ligne épaisse) et avec l'ajout de contraintes dynamiques (ligne trés épaisse).*

6.4 Exemple d'applications

Comme illustration test, prenons le problème d'estimation paramétrique du livre [51], page 165. Les équations de sortie sont données par :

$$y_m(\mathbf{p}, t) = 20\exp(-p_1 t) - 8\exp(-p_2 t). \tag{6.16}$$

Dix mesures ont été collectées sur ce système. Les pavés de la Figure 6-4 représentent les incertitudes associées à chacune des paires de variables de sortie y_i et de mesure du temps t_i, $i \in \{1, \ldots, 10\}$.

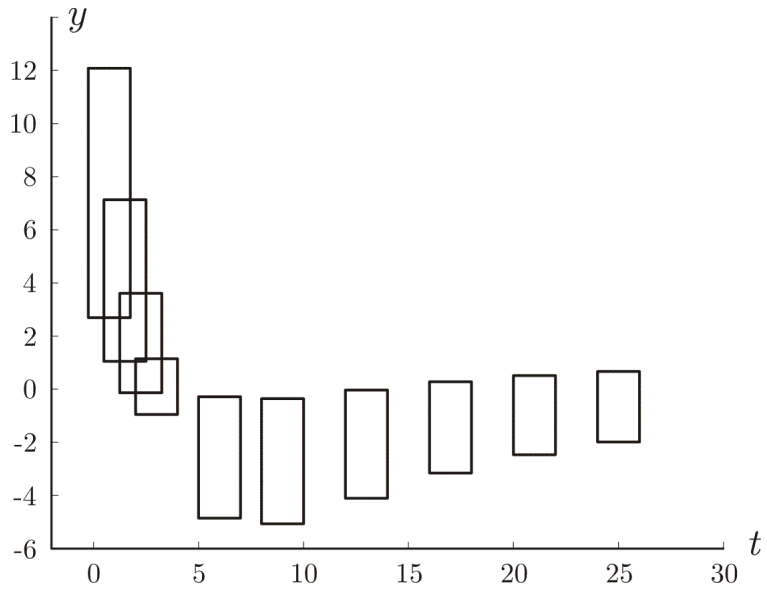

FIG. *6-4 – Représentation des barres d'incertitude. Nous recherchons les paramètres des modèles passant par ces dix barres.*

Les domaines intervalles associés à t_i et y_i sont donnés dans la table suivante :

i	$[t_i]$	$[y_i]$
1	$[-0,25; 1,75]$	$[2,7; 12,1]$
2	$[0,5; 2,5]$	$[1,04; 7,14]$
3	$[1,25; 3,25]$	$[-0,13; 3,61]$
4	$[2; 4]$	$[-0,95; 1,15]$
5	$[5; 7]$	$[-4,85; -0,29]$
6	$[8; 10]$	$[-5,06; -0,36]$
7	$[12; 14]$	$[-4,1; -0,04]$
8	$[16; 18]$	$[-3,16; 0,3]$
9	$[20; 22]$	$[-2,5; 0,51]$
10	$[24; 26]$	$[-2; 0,67]$

L'ensemble des paramètres solutions est défini par :

$$\mathbb{S} = \{\mathbf{p} \in \mathbb{R}^2 \mid \exists t_1 \in [t_1], \ldots, \exists t_{10} \in [t_{10}] \mid y_m(\mathbf{p}, t_1) \in [y_1], \ldots, y_m(\mathbf{p}, t_{10}) \in [y_{10}]\}. \quad (6.17)$$

Une approximation intérieure et extérieure de $\mathbb{S}$ peut être obtenue avec des algorithmes efficaces [51] dédiés à ce problème particulier. Ici, notre but n'est pas de développer un algorithme plus efficace pour ce problème mais de comparer les différentes procédures de contraction. Pour cela nous utilisons un algorithme basique de contraction/bissection (appelé branch and prune algorithms) pour obtenir une approximation extérieure de l'ensemble $\mathbb{S}$. Le nombre des multi-occurrences des variables p_1, p_2 et t_i étant faible dans l'expression des contraintes, la propagation de contraintes

classique se comporte très bien et la plupart des contractions sont optimales. Maintenant si nous réécrivons le modèle correspondant à la sortie du système de la façon suivante :

$$y_m(\mathbf{p}, t) = 20e^{-p_1 t} - 8e^{-p_2 t} + 2t - t - t + \sin^2(t) + \cos^2(t) - 1, \tag{6.18}$$

la procédure de contraction classique, qui est très sensible aux multi-occurrences des variables, devient totalement inefficace. La Figure 6-5 représente le résultat de l'algorithme de bissec-

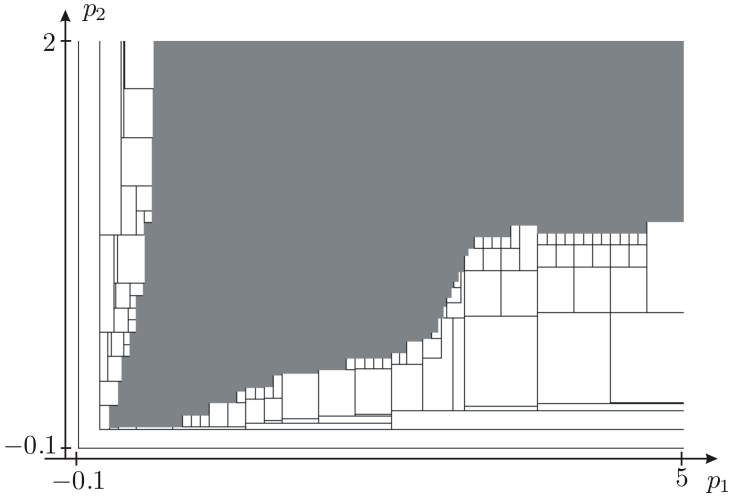

FIG. 6-5 – Caractérisation extérieure de $\mathbb{S}$ obtenue en utilisant un algorithme de bissection/contraction. La contraction a été effectuée en utilisant la propagation de contraintes classique.

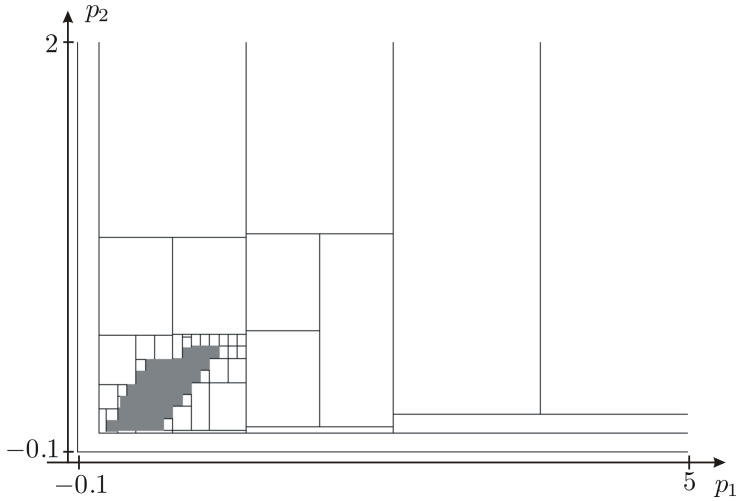

FIG. 6-6 – Caractérisation extérieure de $\mathbb{S}$ obtenue en utilisant un algorithme de bissection/contraction. Ici les contraintes dynamiques basées sur la dérivée ont été ajoutées.

tion/contraction où la contraction est basée sur la propagation de contraintes sur les intervalles.

Pour les pavés blancs, il a été prouvé que leur intersection avec $\mathbb{S}$ est vide. La Figure 6-6 illustre le résultat obtenu avec le même algorithme de bissection/contraction, mais sur cet exemple, les contraintes dynamiques des équations (6.6) ont été utilisées pour les contractions. Cette comparaison démontre que l'ajout de contraintes dynamiques basées sur les dérivées peut réduire de façon efficace le nombre de découpages à effectuer, particulièrement quand les multi-occurrences sont présentes dans l'expression des contraintes.

6.5 Retour sur l'étalonnage des robots

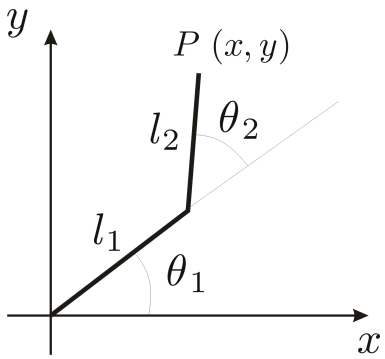

FIG. 6-7 – Schéma d'un robot SCARA. Les angles θ_1 et θ_2 sont les angles des articulations motorisées du robot. l_1 et l_2 sont les paramètres géométriques du robot.

Prenons l'exemple simple de l'étalonnage d'un robot de type SCARA. Ce robot plan, illustré sur la figure 6-7, possède deux axes rotoïdes. Son modèle géométrique direct est le suivant :

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2), \tag{6.19}$$
$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2). \tag{6.20}$$

Avec un offset sur les angles, les équations sont modifiées de la façon suivante :

$$x = l_1 \cos(\theta_1 + \theta_1^{off}) + l_2 \cos(\theta_1 + \theta_2 + \theta_1^{off} + \theta_2^{off}), \tag{6.21}$$
$$y = l_1 \sin(\theta_1 + \theta_1^{off}) + l_2 \sin(\theta_1 + \theta_2 + \theta_1^{off} + \theta_2^{off}), \tag{6.22}$$

Avec le vecteur $\mathbf{x} = (x, y)$ correspondant à la position de l'organe terminal, $\boldsymbol{\theta} = (\theta_1, \theta_2)$ les angles permettant de commander le robot et $\mathbf{p} = (l_1, l_2, \theta_1^{off}, \theta_2^{off})$ les paramètres du modèle à estimer.

Dans ce problème, la dimension nous permet de caractériser l'ensemble des solutions avec des algorithmes de bissection. Ici l'usage de la bissection, que nous nous sommes interdit, permet de représenter un sous-pavage de l'ensemble de solutions et de pouvoir situer la boîte que nous obtenons par rapport à la boîte optimale.

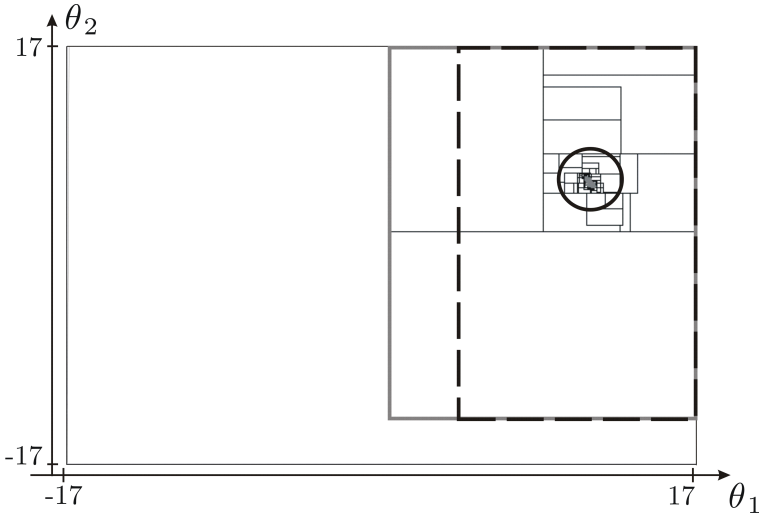

FIG. 6-8 – Présentation du sous-pavage réalisé avec un algorithme de type SIVIA pour le problème de l'étalonnage du robot SCARA avec l'offset.

L'ensemble de solutions pour l'étalonnage du robot SCARA généré par un algorithme de découpage est représenté sur la figure 6-8. L'ensemble de solutions apparaît en gris au centre du cercle. La boîte obtenue par simple propagation a été ajoutée en gris sur la figure, celle obtenue par ajout de contraintes dynamiques utilisant la dérivée apparaît en pointillé noir. Malgré une diminution de la taille de la boîte encadrant l'ensemble de solutions, la boîte optimale n'est pas atteinte même pour ce problème de petite dimension.

En appliquant le principe des CSP dynamiques à notre problème d'étalonnage pour un robot à six degrés de liberté, la boîte obtenue n'est pas plus petite que celle obtenue avec propagation simple. Ce constat d'échec sur ce problème peut s'expliquer par deux choses : i) les expressions matricielles font apparaître des multi-occurrences pour toutes les variables. Le problème étant de grande dimension, le nombre de multi-occurrences est considérable et supprimer la multi-occurrence sur une variable (avec les CSP dynamiques) ne suffit pas pour contracter cette variable. ii) la méthode des CSP dynamiques n'utilise qu'une seule contrainte pour contracter une variable. En effet en réduisant l'influence des multi-occurrences d'une contrainte, nous supprimons des valeurs non consistantes du domaine des variables. Mais ici le problème comporte de nombreuses contraintes et les traiter séparément ne permet pas d'obtenir la consistance globale.

Les CSP dynamiques permettent donc des avancées sur certains problèmes comme celui présenté section 6.4, mais pas sur le problème de l'étalonnage. La suppression analytique des multi-occurrences permettrait sans doute une amélioration des résultats de cette méthode mais elle nous semble encore limitée. En effet ces méthodes ne traitent qu'une contrainte à la fois. L'obtention de la boîte minimale pour chacune des contraintes n'empêchera pas le problème de consistance locale. Un traitement simultané de plusieurs contraintes est donc à trouver pour permettre de lever le problème de consistance locale.

Chapitre 7

Conclusion

Les travaux de cette thèse portent sur l'implémentation et la mise au point de méthodes par intervalles pour des applications en robotique et en automatique. L'application de ces méthodes à l'étalonnage des robots séries a guidé notre travail de recherche. Les difficultés alors rencontrées nous ont orienté vers des améliorations des méthodes existantes. Le détail de ces contributions est résumé avec leurs avantages et inconvénients dans les parties 7.1, 7.2 et 7.3 retraçant ainsi l'évolution pendant ce travail de thèse.

L'étalonnage géométrique de l'ensemble des paramètres du robot n'ayant pas pu être effectué, le problème reste un problème ouvert pour les méthodes par intervalles. La possibilité de réduire le problème de consistance locale dû à la multi-occurrence des variables dans les contraintes ayant été étudiée, il faut trouver d'autres méthodes pour s'approcher de la boîte minimale. Le traitement de plusieurs contraintes de façon simultanée avec une approche matricielle est l'une des perspectives de recherche (voir partie 7.4).

7.1 Amélioration du temps de calcul

7.1.1 Avantages

La première contribution est l'utilisation d'un graphe acyclique orienté pour la représentation des contraintes en machine pour le traitement rapide des problèmes de grandes dimensions. Pour une contrainte du type :

$$\mathbf{y} = \mathbf{f}(\mathbf{x}), \tag{7.1}$$

la propagation rétro-propagation sur le DAG trouve le pavé d'équilibre en diminuant le temps de résolution avec une propagation sur les arbres d'un facteur 10. Bien sûr, une comparaison reste à faire avec les méthodes de réveil de contraintes car celles-ci peuvent aussi s'appliquer à la structure de DAG.

7.1.2 Inconvénients

L'un des inconvénients de cette formalisation est l'obligation pour l'utilisateur de faire un choix des variables d'entrée et de sortie associées à son problème. Il n'y a donc pas, comme pour les algorithmes de réveil de contraintes, de structure dynamique s'adaptant au CSP donné par l'utilisateur.

L'autre inconvénient est le temps de création du DAG avant la propagation rétro-propagation. Cette création de la structure du DAG peut être réalisée une fois pour toute avant la propagation. Si les utilisateurs modifient uniquement les domaines de recherche associés aux variables du CSP alors la phase de compilation n'est pas nécessaire.

7.1.3 Problèmes ouverts

Le problème de la consistance locale qu'il soit dû à la multi-occurrence des variables ou à la présence de plusieurs contraintes n'est pas résolu par la mise en place des DAG. Seul le paramètre temps de calcul est modifié. Deux pistes ont alors été explorées : l'ajout de contraintes redondantes et l'ajout de contraintes dynamiques pour augmenter la contraction obtenue.

7.2 Amélioration de la contraction par l'ajout de contraintes redondantes

L'étalonnage géométrique des robots séries peut être modélisé grâce à une suite de rotations et de translations qui permettent de prendre en compte chaque corps et chaque articulation. Avoir pour la rotation les plus petits domaines associés aux variables devient alors un élément important pour la propagation de contraintes.

L'ajout de contraintes redondantes relatives à la rotation est alors une approche possible pour approcher la boîte optimale. Mais cette approche n'a pas pu améliorer significativement la contraction pour le problème de l'étalonnage avec tous les paramètres. En effet, même si la rotation de la forme :

$$\mathbf{x}_2 = \mathbf{R}_z(\theta).\mathbf{x}_1 \tag{7.2}$$

permet l'obtention de la boîte optimale, elle ne garantit en rien l'obtention de celle-ci lorsque plusieurs rotations s'enchaînent comme dans l'expression :

$$\mathbf{x}_3 = \mathbf{R}_x(\alpha).\mathbf{R}_z(\theta).\mathbf{x}_1. \tag{7.3}$$

De même lorsque les rotations sont couplées avec des translations :

$$\mathbf{x}_3 = \mathbf{R}_x(\alpha).\mathbf{T}_x(d).\mathbf{R}_z(\theta).\mathbf{T}_z(r).\mathbf{x}_1,$$

la boite optimale n'est pas obtenue.

Le traitement des multi-occurrences engendrées par la composition des rotations par des méthodes de consistance plus fortes est donc l'étape suivante que nous nous somme fixée.

7.3 Traitement avec des méthodes de consistance plus fortes

L'apport de l'information contenue dans la dérivée des contraintes du CSP permet de renforcer la contraction des domaines vers la boîte optimale. En effet, certaines parties des domaines intervalles obtenus après simple propagation ne sont pas box consistant (voir chapitre 3 page 49). Des méthodes de réduction permettent alors d'atteindre la propriété de box consistance (voir [70], [10]). Nous avons proposé d'ajouter de façon dynamique aux contraintes liées aux dérivées partielles des contraintes pour renforcer la contraction et résoudre le problème de la multi-occurrence.

La méthode s'affranchit de la multi-occurrence selon une variable mais est toujours limitée par la multi-occurrence des autres variables. Par conséquent, lorsque le problème présente des multi-occurrences regroupées comme dans la contrainte suivante :

$$y = x_1.(1 + \sin(x_1) - \sin^2(x_1)) + x_2.(1 + \sin(x_2) - \sin^2(x_2)), \tag{7.4}$$

l'approche de box consistance aura une importance décisive sur la taille des pavés obtenus. Mais dans le cas des problèmes où les occurrences des variables sont imbriquées, cet avantage est moindre. Dans l'étalonnage du robot STAUBLI, les multi-occurrences sont trop imbriquées pour permettre d'obtenir une amélioration significative. D'autres méthodes doivent donc être trouvées pour vaincre la consistance locale.

7.4 Perspective : Traitement matriciel des contraintes

Jusqu'ici, le problème de l'étalonnage se formalisait comme une succession de rotations et de translations. A l'issue de cette thèse deux voies d'amélioration de la contraction ont été explorées : l'obtention de la contraction optimale pour chaque rotation prise séparément et un traitement des contraintes scalaires visant à réduire l'influence des multi-occurrences. Malgré ces améliorations, la contraction des domaines de l'ensemble des paramètres du problème de l'étalonnage du robot série STAUBLI rx90 n'a pas permis d'obtenir des solutions exploitables en pratique.

7.4.1 Présentation du problème

Pour comprendre, prenons l'exemple de trois rotations successives dans le plan afin de visualiser le phénomène d'enveloppement. Le pavé $[\mathbf{x}]$ sur la figure 7-1 subit trois rotations d'angle θ. A

chaque rotation, l'ensemble des solutions est encadré par un pavé. Sur la figure 7-1, la rotation étant d'un angle de $\frac{\pi}{4}$, l'aire du pavé double à chaque rotation.

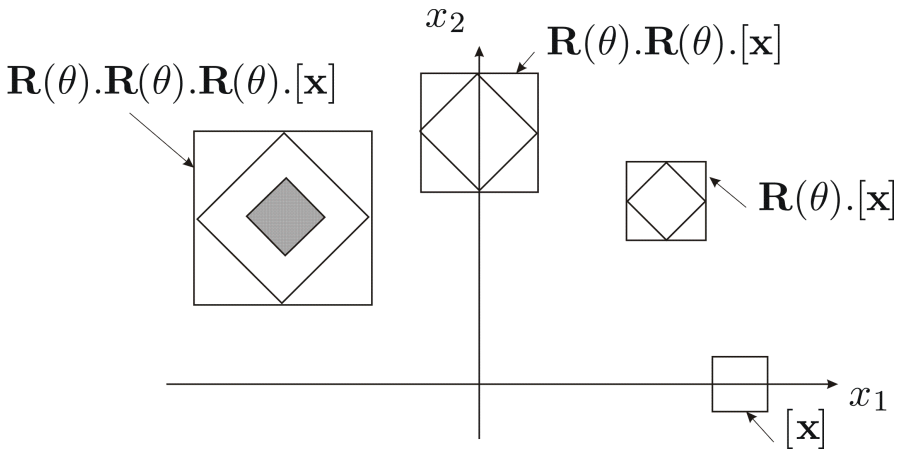

FIG. 7-1 – Illustration du phénomène d'enveloppement dans le cas d'une succession de rotations $\mathbf{R}(\theta)$ d'angle θ.

Le phénomène d'enveloppement (ou wrapping effect) observé ici pour trois rotations est déjà très important. Dans le problème de l'étalonnage où le nombre de rotation est de 12 , ce phénomène devient prépondérant.

Sur la figure 7-1 , le carré gris correspond à l'image du pavé $[\mathbf{x}]$ par la rotation équivalente à 3θ. Dans le cas de l'étalonnage, une possibilité pourrait être de travailler sur le polygone enfermant les solutions en lui appliquant les translations et rotations. Mais nous sommes ici dans un cas de rotation non ponctuelle avec des variables qui prennent leurs valeurs dans des intervalles. Le pavé $[\mathbf{x}]$ après avoir subi une rotation n'a alors plus la forme d'un polygone régulier comme l'illustre la figure 7-2. Ceci rend alors la manipulation de ces ensembles délicate.

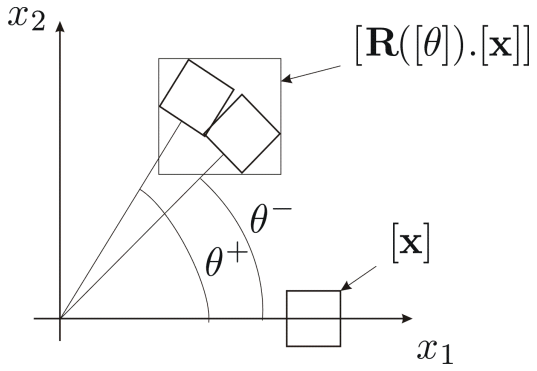

FIG. 7-2 – Ilustration de la rotation d'un angle θ prenant ses valeurs dans un intervalle non ponctuel.

7.4.2 La méthode matricielle de Lohner

Le principe de la méthode proposé par Lohner [50] dans le cadre des équations aux récurrences est le suivant. Pour une équation matricielle :

$$\mathbf{y} = \mathbf{A}_1.\mathbf{A}_2.\mathbf{A}_3.\mathbf{x}, \tag{7.5}$$

sur chacune des matrices $\mathbf{A}_i$ sont appliquées des matrices de transformation ponctuelles permettant de se déplacer dans un repère minimisant l'effet d'enveloppement autour des images de $[\mathbf{x}]$. Dans l'algorithme présenté par Lohner, les matrices de transformation sont construites à partir du centre des matrices $[\mathbf{A}_i]$ et de leurs inverses. L'algorithme prend la forme suivante :

Algorithme Lohner	
entrées : $[\mathbf{x}]$, $[\mathbf{A}_i]$, $\mathbf{C}_i = centre([\mathbf{A}_i])$, $\mathbf{C}_i^{-1} = centre([\mathbf{A}_i])^{-1}$. sortie : $[\mathbf{y}]$.	
1	$\mathbf{B} := \mathbf{Id}$; $\mathbf{B}^{-1} := \mathbf{Id}$;
2	$[\mathbf{r}] := [\mathbf{x}]$;
3	for $i = 1$ to $i = n$ do
4	$\mathbf{B}^{-1} := \mathbf{B}^{-1}.\mathbf{C}_i^{-1}$;
5	$[\mathbf{r}] := \mathbf{B}^{-1}.[\mathbf{A}_i]\mathbf{B}.[\mathbf{r}]$;
6	$\mathbf{B} := \mathbf{C}_i.\mathbf{B}$;
7	$[\mathbf{y}] := [\mathbf{y}] \cap (\mathbf{B}.[\mathbf{r}])$.

Dans notre cas, les matrices $\mathbf{A}_i$ correspondent à des matrices de rotation et de translation dont nous connaissons analytiquement les inverses :

$$(\mathbf{R}(\theta))^{-1} = \mathbf{R}(-\theta), \tag{7.6}$$

pour la rotation et :

$$(\mathbf{T}(d))^{-1} = \mathbf{T}(-d), \tag{7.7}$$

pour la translation.

Faire un essai de cette méthode sur l'étalonnage des robots séries semble une perspective intéressante. La possibilité de coupler la propagation des contraintes avec ces méthodes matricielles permettrait alors sans doute de lever certains pavés d'équilibre.

Les différentes perspectives énoncées dans cette partie sont en cours d'étude, elles correspondront à notre axe de travail à venir.

Annexe A

Implémentation d'IntervalPeeler

Les codes sources des algorithmes de résolution utilisés dans IntervalPeeler sont disponibles à l'adresse http ://www.istia.univ-angers.fr/~baguenar/interval_peeler_console_sources.zip.

La programmation a été réalisée en C++ sur C++Builder à partir des classes crées par Luc Jaulin pour l'analyse par intervalles qui ont été modifiées avec Massa DAO pour ajouter les méthodes de contraction pour les contraintes primitives. Le détail des fichiers est le suivant

Pour l'analyse syntaxique :

Flex_lexer.l fichier de règles lexicales (pour flex)

Bison_parser.y fichier de règles syntaxiques (pour bison)

parser_types.h fichier qui définit la structure des variables et des contraintes lors de la création des arbres pour les contraintes.

InitTree1.h et **InitTree1.cpp** algorithmes de création des arbres.

Pour la conversion de l'arbre en DAG :

TreeToDag1.h et **TreeToDag1.cpp** algorithmes de création du DAG à partir des arbres.

Pour les algorithmes de propagation de contraintes :

dag.h et **dag.cpp** pour la structure du DAG

contractor_test.h et **contractor_test.cpp** pour les algorithmes de contraction

Pour la bibliothèque intervalle :

interval.h et **interval.cpp** pour le codage de l'analyse par intervalles et de la contraction des contraintes primitives.

box.h et **box.cpp** pour les pavés où boites de $\mathbb{R}^n$.

Dans l'ensemble de ces fichiers, les trois fichiers relatifs à l'analyse syntaxique sont présentés dans cette annexe afin de pouvoir rapidement comprendre la syntaxe d'IntervalPeeler.

Syntaxe d'IntervalPeeler : fichier flex

```
%{
////////////////////////////////////////////////
//////code insert at the head of lexer.c////////
////////////////////////////////////////////////
#include <iostream.h>
#include "parser_types.h"
#include "parser_log.h"
////////////////////////////////////////////////
string int2str(int a)
{
AnsiString temp=IntToStr(a);
string b=temp.c_str();
return b;
}
////////////////////////////////////////////////
%}
%option never-interactive
BLANCS        [ \t]+
CHIFFRE       [0-9]
ENTIER        {CHIFFRE}+
EXPOSANT      ([eE][+-]?{ENTIER})
REEL          {ENTIER}("."{ENTIER})?{EXPOSANT}?
LETTER        [A-Za-z]
VARIABLE      {LETTER}({LETTER}|{CHIFFRE})*
MARK   "EndOfFile"|"End"|"Constraints"|"Output variables"|"Input variables"
                            |"Output Variables"|"Input Variables"|"Variables"
%%
{BLANCS}   {
LOGOUT<<nbctr<<" : Flex--> blanc"<<endl;
}
([Ii]"nteger")|"INTEGER"   {
LOGOUT<<nbctr<<" : Flex int"<<endl;
yylval.type='z' ;
return(UNARY_CTR);
}
([Bb]"oolean")|"BOOLEAN"   {
LOGOUT<<nbctr<<" : Flex bool"<<endl;
```

```
yylval.type= 'b';
return(UNARY_CTR);
}
"^"{ENTIER}   {
LOGOUT<<nbctr<<" : Flex--> power: "<<yytext<<endl;
yytext++;
yylval.indice=atoi(yytext) ;
return(POWER);
}
([Ss]"in")|"SIN"   {
LOGOUT<<nbctr<<" : Flex--> sin"<<endl;
yylval.type='S' ; return(UNARY_FCT);
}
([Ss]"in"[cC])|"SINC"   {
LOGOUT<<nbctr<<" : Flex--> sinC"<<endl;
yylval.type='s' ; return(UNARY_FCT);
}
([Cc]"os")|"COS"      {
LOGOUT<<nbctr<<" : Flex--> cos"<<endl;
yylval.type='C' ; return(UNARY_FCT);
}
([Tt]"an")|"TAN"      {
LOGOUT<<nbctr<<" : Flex--> tan"<<endl;
yylval.type='T' ; return(UNARY_FCT);
}
([Aa]"tan")|"ATAN"      {
LOGOUT<<nbctr<<" : Flex--> atan"<<endl;
yylval.type='A' ; return(UNARY_FCT);
}
([Ee]"xp")|"EXP"      {
LOGOUT<<nbctr<<" : Flex--> exp"<<endl;
yylval.type='E' ; return(UNARY_FCT);
}
([Ll]"n")|"LN"         {
LOGOUT<<nbctr<<" : Flex--> ln"<<endl;
yylval.type='L' ; return(UNARY_FCT);
}
([Ss]"qrt")|"SQRT"      {
LOGOUT<<nbctr<<" : Flex--> sqrt"<<endl;
yylval.type='R' ; return(UNARY_FCT);
}
([Ss]"qr")|"SQR"      {
```

```
LOGOUT<<nbctr<<" : Flex--> sqr"<<endl;
yylval.type='2' ; return(UNARY_FCT);
}
([Aa]"bs")|"ABS"      {
LOGOUT<<nbctr<<" : Flex--> abs"<<endl;
yylval.type='B' ; return(UNARY_FCT);
}
([Mm]"ax")|"MAX"      {
LOGOUT<<nbctr<<" : Flex--> max"<<endl;
yylval.type='M' ; return(BINARY_FCT);
}
([Mm]"in")|"MIN"      {
LOGOUT<<nbctr<<" : Flex--> min"<<endl;
yylval.type='m' ; return(BINARY_FCT);
}
([Aa]"ngle")|"ANGLE"   {
LOGOUT<<nbctr<<" : Flex--> angle"<<endl;
return(ANGLE_CTR);
}
"oo" {
LOGOUT<<nbctr<<" : Flex-->"<<yytext<<endl;
return(BIGNUMBER);
}
"pi" {
LOGOUT<<nbctr<<" : Flex-->"<<yytext<<endl;
return(PI);
}
"in" {
LOGOUT<<nbctr<<" : Flex-->"<<yytext<<endl;
return(DOM);
}
{MARK} {
LOGOUT<<nbctr<<" : Flex-->"<<yytext<<endl;
typevar++;
return(DEFINE);
}
{REEL}      {
LOGOUT<<nbctr<<" : Flex--> real: "<<yytext<<endl;
strcpy(yylval.Name,yytext);
yylval.c=atof(yytext) ;
return(NUMBER);
}
```

```
{VARIABLE}   {
LOGOUT<<nbctr<<" : Flex--> variable : "<<yytext<<endl;
strcpy(yylval.Name,yytext);
return(VARIABLE);
}
[-+*\/(),=\[\];^]   {
LOGOUT<<nbctr<<" : Flex-->"<<yytext<<endl;
return *yytext;
}
"//"      {
LOGOUT<<nbctr<<" : Flex--> comment : "<<yytext<<endl;
return(0);
}
"\n"      {
LOGOUT<<nbctr<<" : Flex--> end of line"<<endl;
return(0);
}
.      {
LOGOUT<<nbctr<<" : Flex--> Lexical error"<<endl;
error=true;
char a[1000]; strcpy(a,yytext);
comment=comment+"line : "+int2str(nbctr)+" forbidden Character "+a+"\n";
}
%%
int yywrap(void) {
   return 1 ;
}
```

Syntaxe d'IntervalPeeler : fichier bison

```
%{
////////////////////////////////////////////////
//////code insert at the head of parser.c////////
////////////////////////////////////////////////
#include "parser_types.h"
#include "lexer.c"
#include <stdlib.h>
//ajout17/09/2003
#include "hash_table.cpp"
extern HashTable H;
////////////////////////////////////////////////
int yyerror(char *s)
```

```
{
error=true;
comment=comment+"line : "+int2str(nbctr)+" Grammatical error \n";
printf("ERREUR: %s\n", s); return(0);
}
////////////////////////////////////////////////
void TestMinMax(int minCode)
{
if (Constraint[minCode].Code=='M')
 {
 Constraint[minCode].Code='m';
 TestMinMax(Constraint[minCode].Left_son);
 }
}
////////////////////////////////////////////////
int NewVar(char name[])
{
LOGOUT<<nbctr<<" -->--> Bison: NewVar\n";
char a[50]; //strcpy(a,name);
string NameVar=name;
int i=in(NameVar,H);
int t=V.size();
LOGOUT<<nbctr<<" -->-->--> Bison: NewVar : indice variable "<<i;
LOGOUT<<" size variable"<<t;
if (i!=-1)
 return(i);
else {
 insert(NameVar,t,H);
//renseigne la table de hachage : on a ajoute "NameVar" dans V avec indice t
 V.push_back(Variable( NameVar,typevar ));
 return (t);
 }
}
////////////////////////////////////////////////
int Codage(char Code,int G,int D)
{
int Indice=-1;
LOGOUT<<nbctr<<" -->--> Bison: Codage Expression : "<<Code<<endl;
Constraint.push_back(NodeTree(Code,G,D));
Indice=Constraint.size()-1;
Constraint[G].Father=Indice;
if (D!=-1)
```

```
 Constraint[D].Father=Indice;
return Indice;
}
////////////////////////////////////////////
int Codage(char Code,int x1,int y1,int t,int x2,int y2)
{
int Indice=-1;
LOGOUT<<nbctr<<" -->--> Bison: Codage Angle: "<<Code<<endl;
Constraint.push_back(NodeTree(Code,x1,y1));
Constraint.push_back(NodeTree(Code,x2,y2));
Constraint.push_back(NodeTree(Code,t,-1));
Indice=Constraint.size()-1;
Constraint[x1].Father=Indice;
Constraint[y1].Father=Indice;
Constraint[x2].Father=Indice;
Constraint[y2].Father=Indice;
Constraint[t].Father=Indice;
return Indice;
}
////////////////////////////////////////////////
%}
////////////////////////////////////////////////
//////token definition/////////////////////////
////////////////////////////////////////////////
%token '=' ',' ';'
%token UNARY_CTR ANGLE_CTR
%token BINARY_FCT UNARY_FCT
%token   VARIABLE
%token   NUMBER
%token   '+' '-' '*' '/'
%token   '(' ')' '[' ']'
%token   DOM PI BIGNUMBER
%token   DEFINE
////////////////////////////////////////////////
//////property definition/////////////////////
////////////////////////////////////////////////
%left      '+' '-'
%left      '*' '/'
%left      NEG
%nonassoc   POWER
////////////////////////////////////////////////
//////rules definition/////////////////////////
```

```
////////////////////////////////////////////////
%start Input
%%
Input:
 /* Vide */
 | DEFINE {
LOGOUT<<nbctr<<" -->--> Bison input Comment"<<endl;
}
 | DomainInterval {
LOGOUT<<nbctr<<" -->--> Bison input interval"<<endl;
}
   | Constraint {
LOGOUT<<nbctr<<" -->--> Bison input constraint"<<endl;
Constraint[Constraint.size()-1].Indice=nbctr;
}
 ;
DomainInterval:
 VARIABLE DOM Interval {
LOGOUT<<nbctr<<" -->--> Bison DomainInterval"<<endl;
int i=NewVar($1.Name);
V[i].INT=INITVAR;
if (typevar==-oo) typevar=i;
}
 ;
Interval:
 '[' Reals ',' Reals ']' {
LOGOUT<<nbctr<<" -->--> Bison interval REAL"<<endl;
INITVAR= interval($2.c,$4.c);
}
 |'[' Reals ']' {
LOGOUT<<nbctr<<" -->--> Bison interval REALcst"<<endl;
INITVAR= interval($2.c,$2.c);
}
 | Reals {
LOGOUT<<nbctr<<" -->--> Bison interval REALcst"<<endl;
INITVAR= interval($1.c,$1.c);
}
 | '[' '-' BIGNUMBER ',' '+' BIGNUMBER ']' {
LOGOUT<<nbctr<<" -->--> Bison interval INF"<<endl;
INITVAR= interval(-oo,oo);
}
 | '[' '-' BIGNUMBER ',' Reals ']' {
```

```
LOGOUT<<nbctr<<" -->--> Bison interval INF Real"<<endl;
INITVAR= interval(-oo,$5.c);
}
 | '[' Reals ',' '+' BIGNUMBER ']' {
LOGOUT<<nbctr<<" -->--> Bison interval Real INF"<<endl;
INITVAR= interval($2.c,oo);
}
 ;
Reals:
 NUMBER {
LOGOUT<<nbctr<<" -->--> Bison real pos"<<endl;
$$=$1;
}
 | '-' NUMBER {
LOGOUT<<nbctr<<" -->--> Bison real neg"<<endl;
$2.c= - $2.c ;
$$=$2;
}
 ;
Constraint:
 Expression '=' Expression {
LOGOUT<<nbctr<<" -->--> Bison Constraint"<<endl;
Codage('=',$1.indice, $3.indice);
}
 | UNARY_CTR '(' Expression ')' {
LOGOUT<<nbctr<<" -->--> Bison input UNARY_CTR"<<endl;
Codage($1.type,$3.indice,-1);
}
 | ANGLE_CTR '(' Expression ',' Expression ',' Expression ','
                                    Expression ',' Expression ')' {
LOGOUT<<nbctr<<" -->--> Bison input ANGLE_CTR"<<endl;
Codage('a',$3.indice,$5.indice,$7.indice,$9.indice,$11.indice);
}
 ;
Expression:
 VARIABLE {
LOGOUT<<nbctr<<" --> Yacc: Expression : variable : "<<$1.Name<<endl;
int i=NewVar($1.Name);
Constraint.push_back(NodeTree('V',-1,-1,i));
$$.indice=Constraint.size()-1;
V[i].Occur.push_back($$.indice);
}
```

```
 | NUMBER {
LOGOUT<<nbctr<<" --> Yacc: Expression : number : "<<$1.c<<endl;
Cst.push_back($1.c);
Constraint.push_back(NodeTree('N',-1,-1,666));
$$.indice=Constraint.size()-1;
}
 | Expression '+' Expression {
LOGOUT<<nbctr<<" -->--> Bison +"<<endl;
$$.indice=Codage('+',$1.indice,$3.indice);
}
 | Expression '-' Expression {
LOGOUT<<nbctr<<" -->--> Bison -"<<endl;
$$.indice=Codage('-',$1.indice,$3.indice);
}
 | Expression '*' Expression {
LOGOUT<<nbctr<<" -->--> Bison *"<<endl;
$$.indice=Codage('*',$1.indice,$3.indice);
}
 | Expression '/' Expression {
LOGOUT<<nbctr<<" -->--> Bison /"<<endl;
$$.indice=Codage('/',$1.indice,$3.indice);
}
 | Expression POWER {
LOGOUT<<nbctr<<" -->--> Bison Power"<<endl;
$$.indice=Codage('P',$1.indice,-1);
Constraint[$$.indice].Indice=$2.indice;
}
 | '-' Expression %prec NEG {
LOGOUT<<nbctr<<" -->--> Bison _"<<endl;
$$.indice=Codage('_',$2.indice,-1);
}
 | UNARY_FCT '(' Expression ')' {
LOGOUT<<nbctr<<" -->--> Bison UNARY_FCT"<<endl;
$$.indice=Codage($1.type,$3.indice,-1);
}
   | BINARY_FCT '(' MinOrMaxExp ')' {
LOGOUT<<nbctr<<" -->--> Bison BINARY_FCT"<<endl;
$$=$3;
if ($1.type=='m') TestMinMax($3.indice);
}
 | '(' Expression ')' {
LOGOUT<<nbctr<<" -->--> Bison: Expression : ()"<<endl;
```

```
$$ = $2 ;
}
 ;
MinOrMaxExp :
 Expression ',' Expression {
LOGOUT<<nbctr<<" -->--> Bison BINARY_FCT"<<endl;
$$.indice=Codage('M',$1.indice,$3.indice);
}
 | MinOrMaxExp ',' Expression {
LOGOUT<<nbctr<<" -->--> Bison BINARY_FCT"<<endl;
$$.indice=Codage('M',$1.indice,$3.indice);
}
 ;
%%
```

Structure entre flex et bison

```
#ifndef __NODE__
#define __NODE__

#include <vector.h>
#include <string.h>
#include <deque.h>
#include <fstream.h>
#include "interval.h"

string int2str(int a);

class  NodeTree {
 public:
      char Code;
      int Left_son;
      int Rigth_son;
      int Father;
      int Mark;
      int Indice;
 NodeTree(char c,int l,int r)
   {Father=-1; Code=c; Indice=-1; Left_son=l; Rigth_son=r; Mark=-1; };
 NodeTree(char c,int l,int r,int i)
   {Father=-1; Code=c; Indice=i; Left_son=l; Rigth_son=r; Mark=-1; };

};
```

```
class  Variable{
 public:
       string Name;
      int Code;
        vector <int> Occur ;
        bool Traited ;
        int Mark;
        int Indice;
      interval INT;
 Variable()
   {Name=""; Code=-1; INT=interval(-oo,oo); Traited=false;Mark=-1;};
 Variable(string Name1, int Code1, interval INT1)
   {Name=Name1; Code=Code1; INT=INT1; Traited=false; Mark=-1;};
 Variable(string Name1, int Code1)
   {Name=Name1; Code=Code1; INT=interval(-oo,oo); Traited=false;Mark=-1;};

};

typedef struct Flex2Bison {
                  int indice;
                  char type;
                  double c; //fichier constantes
                  char Name[100];    //variable name
      } flex2bison;        //struct flex -> bison


#define YYSTYPE flex2bison
extern YYSTYPE yylval;

extern fstream LOGOUT ;
extern interval INITVAR;
extern int typevar;
extern bool error;
extern string comment;
extern int nbctr;

extern deque<NodeTree> Constraint;
extern deque<Variable> V;
extern deque<double> Cst;

#endif
```

Bibliographie

[1] `http ://www.cs.utep.edu/interval-comp/`.

[2] `http ://www.ti3.tu-harburg.de/ rump/intlab/`.

[3] Spacesolver, available at, `http ://liawww.epfl.ch/~lottaz/spacesolver/`.

[4] A. Aho, R. Sethi, and J. Ullman. "Compilateurs Principes, techniques et outils". Informatique intelligence artificielle. InterEditions (1995).

[5] G. Ambarish, A. Quaid, and M. Peshkin. Complete parameter identification from partial pose measurement. In "IEEE International Conference on Robotics and Automation" (1993).

[6] N. Andreff, P. Renaud, F. Pierrot, and P. Martinet. Vision-based kinematic calibration of an h4 parallel mechanism : practical accuracies. *Industrial Robot International Journal* **31**(3), 273–283 (2004).

[7] I. A.N.S.I. "A standard for binary floating-point arithmetic". ANSI/IEEE Std. 754-1985, New York (1985).

[8] J. Armengol. "Simulation of dynamical systems withstructured uncertainty : an overview". PhD dissertation, Universitat de Girona (1998).

[9] X. Baguenard, M. Dao, L. Jaulin, and W. Khalil. Méthodes ensemblistes pour l'étalonnage géométrique. *Journal Européen des Systèmes Automatisés* **37**(9), 1059–1074 (2003).

[10] F. Benhamou, F. Goualard, L. Granvilliers, and J. F. Puget. Revising hull and box consistency. In "Proceedings of the International Conference on Logic Programming", pp. 230–244, Las Cruces, NM (1999).

[11] F. Benhamou, D. McAllester, and P. van Hentenryck. CLP (intervals) revisited. In M. Bruynooghe, editor, "Proceedings of the International Logic Programming Symposium", pp. 124–138, Ithaca, NY (1994).

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

[13] J. Berg. Path and orientation accuracy of industrial robots. *International Journal of Advanced Manufacturing Technology* **8**, 29–33 (1993).

[14] R. Bernhardt and S. Albright. "Robot Calibration". ChapmanHall, London (1993).

[15] C. Bessière. Arc-consistency in dynamic constraint satisfaction problems. In "In Proceedings of the 9th AAAI", pp. 221–226 (1991).

[16] I. Braems. "Analyse par intervalles pour l'estimation et la commande robuste". PhD dissertation, Université Paris Sud, Orsay, France (2002).

[17] I. Braems, F. Berthier, L. Jaulin, M. Kieffer, and E. Walter. Guaranteed estimation of electrochemical parameters by set inversion using interval analysis. *Journal of Electroanalytical Chemistry* **495**(1), 1–9 (2001).

[18] G. Calderón-Espinoza, J. Armengol, M. Ángel Sainz, and P. Herrero. Combining interval and qualitative reasoning for fault diagnosis. In "IFAC World Congress" (2005).

[19] J. G. Cleary. Logical arithmetic. *Future Computing Systems* **2**(2), 125–149 (1987).

[20] D. Daney, A. Neumaier, and Y. Papegay. Interval methods for certification of the kinematic calibration of parallel robots. In "IEEE International Conference on Robotics and Automation", New Orleans, LA (2004).

[21] M. Dao, X. Baguenard, and L. Jaulin. "Proj2d, disponible sur `http ://www.istia.univ-angers.fr/~dao/`". LISA, ISTIA, Université d'Angers (2004).

[22] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence* **32**(3), 281–331 (1987).

[23] J. Denavit and R. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics* **17**, 215–221 (1955).

[24] N. Deo. "Graph Theory with Applications to Engineering and Computer Science". Prentice-Hall, Englewood Cliffs, NJ (1974).

[25] R. Dorf and R. Bishop. "Modern Control Systems". Electrical and Computer Engineering : Control Engineering. Addison-Wesley Publishing Compagny (1995).

[26] M. Driels and U. Pathre. Robot calibration using an automatic theodolite. *Journal of Advanced Manufacturing Technology* **9**, 114–125 (1994).

[27] P. Dwyer. Linear computations. *"John Wiley and Sons"* (1951).

[28] L. Everett. Forward calibration of closed loop jointed manipulators. *The Int. J. of Robotics Research* **8**(4), 85–91 (August 1989).

[29] L. Granvilliers. "Consistances locales et transformations symboliques de contraintes d'intervalles". PhD dissertation, Université d'Orléans, France (1998).

[30] L. Granvilliers. On the combination of interval constraint solvers. *Reliable Computing* **7**(6), 467–483 (2001).

[31] L. Granvilliers. "RealPaver, available at , `http ://www.sciences.univ-nantes.fr /info/perso/permanents/granvil/ realpaver/`". IRIN, University of Nantes (2002).

[32] E. R. Hansen. "Global Optimization using Interval Analysis". Marcel Dekker, New York, NY (1992).

[33] J. Hollerbach. A survey of kinematic calibration. *The Robotics Review, MIT Press Cambridge* **1**, 207–242 (1989).

[34] J. Hollerbach and C. Wampler. A taxonomy of kinematic calibration methods. *International Journal of Robotics Research* **14**, 573–591 (1996).

[35] E. Hyvönen. Constraint reasoning based on interval arithmetic ; the tolerance propagation approach. *Artificial Intelligence* **58**(1–3), 71–112 (Dec. 1992).

[36] L. Jaulin. Consistency techniques for the localization of a satellite. In "1st International Workshop on Global Constrained Optimization and Constraint Satisfaction (Cocos'02)", Nice, France (2002).

[37] L. Jaulin, M. Christie, and L. Granvilliers. Quelques applications de la propagation de contraintes sur les domaines continus en automatique. In "Onzièmes Journées Francophones de Programmation Logique et Programmation par Contraintes (JFPLC 2002)", Nice, France (2002).

[38] L. Jaulin and E. Walter. Guaranteed nonlinear estimation and robust stability analysis via set inversion. In "Proceedings of the 2nd European Control Conference", pp. 818–821 (1993).

[39] R. Judd and A. Knasinski. A technique to calibrate industrial robot with experimental verification. *IEEE Transactions on Robotics and Automation* **6**(1), 20–30 (1990).

[40] R. Kearfott and V. Kreinovich. "Applications of Interval Computations". Kluwer, Dordrecht, the Netherlands (1996).

[41] R. B. Kearfott. Interval mathematics techniques for control theory computations. In K. Bowers and J. Lund, editors, "Computation and Control. Proceedings of the Bozeman Conference", vol. 20 of "Progress in Systems and Control Theory", pp. 169–178. Birkhäuser, Boston, MA (1989).

[42] R. B. Kearfott. "Rigorous Global Search : Continuous Problems". Kluwer, Dordrecht, the Netherlands (1996).

[43] W. Khalil and S. Besnard. Comparison study of the geometric parameters calibration methods. *Int. Journal of robotics and Automation* **15**(2), 56– 67 (2000).

[44] W. Khalil and E. Dombre. "Modélisation, identification et commande des robots, Collection Robotique". Hermès, Paris (1999).

[45] W. Khalil and M. Gautier. Calculation of the identifiable parameters for robots calibration. In "9 th IFAC/IFORS Symposium on Identification and System Parameter Estimation", pp. 888–892, Budapest, Hungary (1991).

[46] W. Khalil and J.-F. Kleinfinger. A new geometric notation for open and closed-loop robots. *Conference on robotics and Automation* pp. 1174–1180 (1986).

[47] N. A. Khlebalin. Interval automatic systems - theory, computer-aided design and applications. *Interval Computations* **3**, 106–115 (1992).

[48] L. Kolev. An interval first-order method for robustness analysis. In "Procedings of the IEEE International Symposium on Circuits and Systems", pp. 2522–2524, Chicago, IL (1993).

[49] L. V. Kolev, V. M. Mladenov, and S. S. Vladov. Interval mathematics algorithms for tolerance analysis. *IEEE Transactions on Circuits and Systems* **35**(8), 967–974 (1988).

[50] U. Kulisch, R. Lohner, and A. Facius. "Perspectives on Enclosure Methods". Springer-Verlag, Vienna (2001).

[51] L. Jaulin, M. Kieffer, O. Didrit, E. Walter. "Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics". Springer-Verlag, London (2001).

[52] C. Lottaz. "Collaborative design using solution spaces". PhD dissertation 2119, Swiss Federal Institute of Technology in Lausanne, Switzerland (2000).

[53] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence* **8**(1), 99–118 (Feb. 1977).

[54] S. A. Malan, M. Milanese, and M. Taragna. Robust analysis and design of control systems using interval arithmetics. *Automatica* **33**(7), 1363–1372 (1997).

[55] J. P. Merlet. "Les robots parallèles". Hermès, Paris, France (1990).

[56] J.-P. Merlet. Alias : an interval analysis based library for solving and analyzing system of equations. In "In SEA", Toulouse, France (2000).

[57] R. E. Moore. "Interval Analysis". Prentice-Hall, Englewood Cliffs, NJ (1966).

[58] R. E. Moore. "Methods and Applications of Interval Analysis". SIAM, Philadelphia, PA (1979).

[59] R. E. Moore. Parameter sets for bounded-error data. *Mathematics and Computers in Simulation* **34**(2), 113–119 (1992).

[60] A. Neumaier. "Interval Methods for Systems of Equations". Cambridge University Press, Cambridge, UK (1990).

[61] PrologIA. "Prolog IV constraints inside". Prolog IV reference manual (1996).

[62] J. Puget and M. Lecomte. Beyond the glass box : Constraints as objects. In "Proceedings of ILPS'95 International Logic Programming Symposium", pp. 513–527, Portland, USA (1995).

[63] S. Ratschan. Approximate quantified constraint solving (AQCS). `http ://www.risc.uni-linz.ac.at/research/software/AQCS` (2000). Software package.

[64] H. Schichl and A. Neumaier. Interval analysis on directed acyclic graphs for global optimization. *Journal of Global Optimization* (2004).

[65] J. Shamma and D. Whitney. A method for inverse robot calibration. *Transactions of the ASME Journal of Dynamical Systems Measurement and Control* **109**, 36–43 (1987).

[66] V. Stahl. "Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equation". PhD dissertation, University of Linz, Austria (1995).

[67] T. Sunaga. Theory of interval algebra and its application to numerical analysis. *In RAAG Memoirs Ggujutsu Bunken Fukuy-kai* **2**, 29–46 (1958).

[68] M. van Emden. Algorithmic power from declarative use of redundant constraints. *Constraints* **4**(4), 363–381 (1999).

[69] P. van Hentenryck, Y. Deville, and L. Michel. "Numerica : A Modeling Language for Global Optimization". MIT Press, Boston, MA (1997).

[70] P. van Hentenryck, L. Michel, and F. Benhamou. `Newton` : Constraint programming over nonlinear constraints. *Science of Computer Programming* **30**(1–2), 83–118 (Jan. 1998).

[71] P. van Hentenryck, L. Michel, and Y. Deville. "Numerica - A Modelling Language for Global Optimization". MIT Press, Cambridge, Massachusetts (1997).

[72] M. Vincze, J. Prenninger, and H. Gander. A laser tracking system to measure position orientation of robot end effectors under motion. *International Journal of Robotics Research* **13**, 305–314 (1994).

[73] D. Waltz. Generating semantic descriptions from drawings of scenes with shadows. In P. H. Winston, editor, "The Psychology of Computer Vision", pp. 19–91. McGraw-Hill, New York, NY (1975).

[74] K. Wang and J. Lienhardt. Robot kinematic calibration using genetic algorithms. In "IPROMS Conference Intelligent Production Machines and systems" (2005).

[75] M. Warmus. Calculus of approximations. *Bulletin de l'Academie Polonaise des Sciences* **5**(4), 253–257 (1956). Available at : `http ://www.cs.utep.edu/interval-comp/warmus.pdf`.

[76] R. Young. The algebra of many-valued quantities. *Mathematische Annalen* **104**, 260–290 (1931). Available at : `http ://www.cs.utep.edu/interval-comp/young.pdf`.

[77] H. Zhuang, J. Wu, and W. Huang. Optimal pmanning of robot calibration experiments by genetic algotithms. In "International Conference on Robotics and Automation", Minneapolis, Minnesota (1996).

Propagation de contraintes sur les intervalles. Application à l'étalonnage des robots.

Xavier BAGUENARD

Résumé : Dans cette thèse, une méthode ensembliste utilisant la propagation de contraintes sur les intervalles est proposée pour l'étalonnage géométrique d'un robot à 6 degrés de libertés. Cette méthode permet d'enfermer chaque paramètre à estimer à l'intérieur d'un intervalle dont la largeur dépend de l'incertitude sur les mesures collectées, des erreurs de modélisation et du pessimisme inhérent aux méthodes ensemblistes.

Le problème de l'étalonnage géométrique est présenté et formalisé comme un problème de satisfaction de contraintes. Mises sous la forme de graphe acyclique orienté, les contraintes sont ensuite projetées sur les domaines afin de determiner un intervalle le plus petit possible pour chacun des paramètres recherchés.

La plus petite boîte encadrant l'ensemble des solutions n'étant pas forcément obtenue, un ajout de contraintes redondantes liées au problème ou aux dérivées des contraintes est proposé afin d'améliorer les résultats obtenus par la propagation.

Mots clés : analyse par intervalles, calcul ensembliste, estimation ensembliste, étalonnage des robots séries, graphe acyclique orienté (DAG), méthodes garanties, problème de satifaction de contraintes (CSP), propagation de contraintes, robotique.

Interval constraints propagation and robots calibration

Xavier BAGUENARD

Abstract : Geometric calibration is used to improve positioning accuracy of industrial robots. This thesis presents a method using interval constraint propagation applied to the kinematic calibration of a six degrees of freedom robot. For each parameter, an interval is computed by the method to enclose the actual value for the parameter in spite of uncertainties on measures, model errors and pessimism.

Geometric calibration problem can be cast into a constraints satisfaction problem involving variables, domains and constraints. These constraints are then projected on the domains for the parameters in order to determine the smallest interval for each variable.

Unfortunately, the smallest box enclosing the solution set is not always obtained. The addition of redundant constraints generated automatically by derivation made it possible to improve the contractions of the domains.

Key-Words : interval analysis, set computation, set estimation, serial robots calibration, directed acyclic graph (DAG), reliable methods, constraints satisfaction problem (CSP), constraints propagation, robotics.