UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS - UFR
Sciences
École Doctorale STIC

THÈSE

pour obtenir le titre de
Docteur en Sciences
de l'Université de Nice-Sophia Antipolis

Discipline: Informatique

présentée et soutenue par **Carlos GRANDÓN**

Résolution de systèmes d'équations de distance avec incertitudes

Thèse dirigée par **Bertrand Neveu**
et préparée à l'INRIA Sophia-Antipolis, projet COPRIN

Soutenue le 22 Mars 2007

Jury:

M. Benhamou, Frédéric	Professeur	Rapporteur
M. Jaulin, Luc	Professeur	Rapporteur
M. Lhomme, Olivier	Principal Scientist	Examinateur
M. Neveu, Bertrand	Ingénieur en Chef P.C.	Directeur de thèse
M. Rueher, Michel	Professeur	Président
Mme. Sam-Haroud, Jamila	Professeur assistant	Examinateur

Abstract

In this thesis we are interested in a particular class of problems which frequently appear in robotics (and many other areas as chemistry, molecular biology, Computer-Aided Design (CAD), and aeronautics). They are systems of distance equations with uncertainties.

Uncertain values mean values which are not exactly determined but are bounded by well-known limits. These values are represented as intervals, and frequently come from measurements. In a model, these values appear as existentially quantified parameters.

Solving such a problem with uncertainties means to find a set of solutions taking into account these inaccuracies in order to obtain *certified* answers (in the way that no solution is lost).

The aim of the works contained in this thesis is to solve systems of distance equations with uncertainties in their parameters as accurately as possible, combining techniques from Constraint Programming and Interval Analysis communities.

A common approximation for the solutions for these types of problems is to replace parameters with interval values by real numbers, and to solve the problem without considering the inaccuracies. We show that this approximation is not convenient, especially when certified solutions are required (for example for safely reasons for a Surgical Robot).

In a first phase, we propose a special Branch and Prune algorithm with conditional bisection which is able to compute a rough approximation of each continuum of solutions for a given problem.

A rough approximation (a box) is not enough in all the cases, thus a sharp approximation (a set of boxes) describing continuous solution sets is often required. We show that this approximation must consider an *inner box test* in order to detect large parts of the search space containing only solutions to the problem. Using inner box tests not only reduces the number of generated boxes but also provides more information about the geometry of the solutions set. We propose and compare various inner box tests for distance equations with uncertainties.

When a single solution point belonging to a continuum of solutions is given, an inner box around this point and totally included within the continuum of solutions may be very interesting for *tolerance issues*. For this reason we propose a strategy for building such a box based on theoretical results of Modal Interval Analysis combined with a well-known technique of Constraint Programming called

projection.

Finally, the developed techniques are illustrated on a real problem of Robotics in which we solve the direct kinematics of a special class of parallel robot.

Résumé

Nous nous intéressons dans le cadre de cette thèse à une classe particulière de problèmes qui apparaissent fréquemment en robotique (et dans beaucoup d'autres domaines comme la chimie, la biologie moléculaire, la conception assistée par ordinateur (CAO), et l'aéronautique). Ces sont les systèmes d'équations de distance avec des incertitudes.

Nous considérons les valeurs entachées d'incertitudes comme des valeurs qui ne sont pas exactement connues mais que l'on peut considérer comme étant dans des limites bien définies. Ces valeurs sont représentées par des intervalles, et représentent fréquemment les mesures de quantités physiques. Dans les modèles, elles peuvent apparaître sous forme de paramètres existentiellement quantifiés.

Résoudre un problème avec des incertitudes signifie trouver l'ensemble des points solutions en considérant les inexactitudes des données, afin d'obtenir des réponses *certifiées* (dans le sens où aucune solution n'est négligée).

Le but des travaux contenus dans cette thèse est de résoudre des systèmes d'équations de distance avec des incertitudes dans leurs paramètres de la manière la plus précise possible, en combinant différentes techniques d'analyse par intervalles et de programmation par contraintes.

Une approximation fréquemment utilisée pour gérer de tels problèmes est de remplacer les paramètres mal connus par des valeurs réelles, et ainsi ne pas prendre en compte les incertitudes. Nous montrons que cette approche n'est pas adaptée, surtout quand les solutions doivent être certifiées (par exemple, pour des raisons de sécurité pour un robot chirugical).

Dans une première phase, nous proposons un algorithme spécifique de type *Branch and Prune* combiné avec une bissection conditionnelle qui permet de calculer une approximation grossière des différents ensembles contenant des continuums de solutions pour un problème donné.

Comme la donnée d'une approximation grossière (une boîte) de chaque continuum de solutions n'est pas suffisante dans tous les cas, il est parfois nécessaire de calculer une approximation plus précise décrivant chaque ensemble continu de valeurs admissibles. Nous montrons que pour calculer cette approximation, il faut considérer un *test de boîte intérieure*, afin de détecter des parties de l'espace contenant seulement des solutions au problème. L'utilisation d'un tel test réduit la quantité de boîtes produites, et fournit plus d'informations à propos des différentes zones solutions. Nous proposons et comparons quelques tests adaptés pour les

équations de distance avec incertitudes.

Étant donné un point solution appartenant à un continuum de solutions, on peut s'intéresser à une boîte autour de ce point qui soit totalement incluse dans le continuum de solutions (pour des raisons de *tolérance*, par exemple). Pour cela nous proposons une stratégie de construction d'une telle boîte basée sur des résultats théoriques sur les intervalles modaux combinés avec une technique connue de programmation par contraintes appelée *projection*.

Finalement, nous illustrons les techniques développées dans cette thèse sur un problème de robotique qui consiste à calculer la position et l'orientation d'un robot parallèle.

Remerciements/Agradecimientos/Acknowledgements

I had the chance of being helped by people speaking many different languages. It is for this reason that I would like to thank each of them in his/her own language.

Tout d'abord, je souhaite remercier chaleureusement Bertrand Neveu, mon directeur de thèse qui a cru en moi depuis le début (mon stage), pour ses grandes qualités personnelles et professionnelles avec lesquels il a su encadrer ma thèse. Sa disponibilité (toujours à 100%) et la qualité de ses remarques ont été un pilier fondamental de ma réussite. Merci Bertrand!

Je tiens à remercier aussi Frédéric Benhamou et Luc Jaulin pour avoir accepté d'être à la fois rapporteurs de ma thèse et membres du jury, et pour la qualité de leurs remarques que j'ai énormément appréciée. Je remercie aussi Olivier Lhomme, Michel Rueher et Jamila Sam-Haroud pour avoir accepté de faire partie de mon jury de thèse.

Ma plus profonde gratitude à tous les membres du projet COPRIN, en particulier à Jean-Pierre Merlet (chef du projet), pour m'avoir accueilli dans son équipe et m'avoir aidé à mieux comprendre plusieurs sujets auxquels je me suis intéressé pendant mes années d'études. Un très grand merci à David Daney, que je considère comme un catalyseur de la recherche au centre du projet. Merci à lui pour faire les liens entre les travaux de différentes personnes et pour favoriser l'échange des idées entre elles.

Je voudrais exprimer ma gratitude à tous mes collègues et amis (Alexandre Goldsztejn, Gilles Chabert, Gilles Trombettoni, Odile Pourtallier, Yves Papegay, Raphaël Pereira...), pour m'avoir supporté et encouragé pendant mes moments de doute et de faiblesse. Merci à tous!

Evidemment, je ne peux pas oublier deux personnes qui m'ont beaucoup aidé à surmonter d'autres difficultés et ont contribué au bon déroulement de ma thèse (et qui n'ont pas forcément de rapport avec la recherche) Vanessa Wallet et Corinne Mangin. J'apprécie tous vos efforts et votre travail.

Una parte muy importante en el éxito de una persona es el apoyo de su familia y amigos. Por esta razón quiero expresar mis más profundos agradecimientos a mi familia: Marcela, Patricio y Valentina, que estuvieron conmigo y compartieron mis altos y bajos a lo largo de mis años de estudio. Gracias por su tiempo y esfuerzo. Sin ustedes, no habría podido llegar a un final exitoso.

Gracias a mis amigos de la *"mafia latinoamericana"*: chilenos, argentinos, mexicanos, colombianos, todos los que me acompañaron y apoyaron durante todos estos años. Mis agradecimientos especiales a Gonzalo, Tomás, Marcelo, Nelson, Tamara, Antonio, Mario y tantos otros que estuvieron siempre ahí cuando tuve algún problema. Muchas gracias muchachos!

To conclude, I would like to thank all the people that have contributed in some way to the good end of this work and that I have not explicitly named above for some reason.

This work was partially supported by the National Commission for Scientific and Technological Research (CONICYT), Chile.

Contents

Chapter 1

Introduction

Many problems in chemistry, robotics, or molecular biology can be modeled as solving a system of equations and inequalities. Although in some particular cases these systems can be *easily solved*, in the general case they are not. Among the most important difficulties that it is necessary to handle are the *non-linearity* of the equations/inequalities and the *uncertainties*.

In this thesis we are interested in a particular class of equations which frequently appear in robotics (and other areas) and are the so-called *Distance Equations* [21].

A distance equation is a relation between two points in a n-dimensional space that constraints their relative position. In this document only Euclidean distances are considered, therefore the expression of this relation between two points $\vec{x}, \vec{y} \in \mathbb{R}^n$ is as follows:

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1.1}$$

where x_i and y_i are the coordinates of $\vec{x}$ and $\vec{y}$ in the Euclidean space, respectively.

A set of points in space constrained by distance equations forms a system of distance equations. Solving such a system means to determine the position of each unknown point, based on the position of the known fixed points and the set of distance equations between them.

Many real-life problems involve these types of systems. Here are some examples in different areas:

Robotics

Solving the kinematics relationship of a robot is a fundamental problem in Robotics and Control [143, 145]. This problem relates the position and orientation of the robot end-effector to the set of the control parameters associated to the robot. Kinematics involves two fundamental questions:

1. *Direct Kinematics*: being given the design parameters of the robot and a set of control input, determine the pose of the end-effector.

2. *Inverse Kinematics*: being given the design parameters of the robot and a pose for the end-effector, find the control input.

The first one is a difficult problem for parallel robots [144] because it involves a system of non-linear equations with (most of the time) several possible solutions. On the other hand, solving the inverse kinematics of a parallel robot is (usually) easier than the direct kinematic, but depends on the mechanical structure of the robot.

A possible formulation for both problems involves a system of distance equations. This system relates a set of parameters (i.e. the coordinates of some specific points of the robot and the lengths of some robot's components) with the position of the end-effector.

Molecular Biology

It is well-known that proteins play an important role in metabolism control. One of the main factors of their effectiveness depends on their spatial structure. For this reason, finding the possible structures of protein (i.e. their *conformation*) is one of the major problems in molecular biology.

One of the methods to determine these structures is based on the resolution of a system of distance equations, the distance being measured with Nuclear Magnetic Resonance (NMR) data. The measurement principle is to irradiate a molecule using different frequencies, and to measure the influence that one resonating nucleus may have on nearby nuclei. As this influence is proportional to the distance between nuclei, a system of distance equations can be obtained [44].

Many strategies have been applied to this problem, and some of them are based on a Constraint Programming approach [8, 127].

Aeronautics

For navigation and safety reasons the pilot of an airplane must always know the altitude of the plane. In order to do it the pilot relies on *redundant* measurements (i.e. different measurement systems are used). A common and well-known measurement system is based on the pressure of the atmosphere (which depends on the altitude). Eventhough this is a good system, it cannot be the only one.

Another interesting measurement system is based on distance information between the plane and some fixed points on the ground [175]. In this system, a radio receiver is installed in the plane, and it is configured for receiving information coming from three or more ground level fixed antennas. This information gives an approximate distance between the plane and each antenna, and it is combined with the known position of the antennas in a system of distance equations.

The same principle is used for Global Positioning System (GPS). A formal method for solving these types of problems (without considering uncertain data) has been proposed in [137].

The previous three examples are only some situations involving distance equations. It is important to notice that in many of them, part of the input data may come from measurements (e.g. the distances in the molecular biology problem) or relies on some a-priori information (e.g. the respective location of the specific parts on the robot). In both cases these values are only approximations of the actual values. We call them *values with uncertainty*.

There are various forms to express these uncertainties. For example, associating a probability distribution to the error, giving several possible values, or simply giving two limits representing their possible minimum and maximum values. In this thesis, the last option has been selected to represent the uncertainty, and the only additional information required is a bounded error for the values of the parameters of a system.

Thus, some parameters of a system are represented by intervals.

The use of intervals for representing uncertainties in the input data can have different meanings, and it is important to specify which is the associated interpretation. For example, solving the simple equation $a \cdot x = b$, where x is the variable and a, b represent parameters with interval values may raise different issues:

- Finding all the values x for which there exist a and b such that $a \cdot x = b$?
- Finding all the values x such that for all values a there exists a value b,

$a \cdot x = b$?

- Finding all the values x such that for all b there exists a value a, $a \cdot x = b$?
- Finding all the values x for which for all a and b, $a \cdot x = b$?

Actually, some of them have a sense which is directly related with the characteristics of the given problem.

1.1 Problem Statement

This thesis is dedicated to the solving of systems of distance equations with uncertainties. Being given a set of points $\vec{x}_p \in \mathbb{R}^n$ $(p = 1, \ldots, n)$ and a set of distances d_{jk} $(j, k \in \{1, \ldots, n\})$ between them, we are interested in the system of m equations:

$$eq_i : \sum_{l=1}^{n} (x_{jl} - x_{kl})^2 = d_{jk}^2 \qquad j, k \in \{1, \ldots, n\};\ i = 1, \ldots, m \tag{1.2}$$

where x_{jl} represents the coordinate l of the point $\vec{x}_j$, and d_{jk}^2 represents the square of the distance between the points $\vec{x}_j$ and $\vec{x}_k$.

Some points and the distances are considered as parameters of the problem. Parameters with uncertainty are represented by intervals, thus a fixed point with uncertainty is represented by an interval vector.

We define a solution of the problem as *an instantiation of the unknown points such that there exists a value of the parameters for which all the equations in the system are verified.*

Example 1.1.1 *Consider the following system of two distance equations:*

$$(x_{11} - x_{21})^2 + (x_{12} - x_{22})^2 = d_{12}^2 \tag{1.3}$$
$$(x_{11} - x_{31})^2 + (x_{12} - x_{32})^2 = d_{13}^2 \tag{1.4}$$

where $\vec{x}_1 = (x_{11}, x_{12})$ *is unknown and* $\vec{x}_2 = (x_{21}, x_{22}), \vec{x}_3 = (x_{31}, x_{32}), d_{12}^2$, *and* d_{13}^2 *are parameters of the system.*

The aim of the works presented in this thesis is to describe the set of all solutions of the problem as accurately as possible. As the set of solutions are not isolated points in the space but continuous solution sets, different strategies to

approximate these solution sets have been developed. These strategies are based on two different but complementary approaches: *Interval Analysis* and *Constraint Programming*.

1.2 Outline of the Document

This document is divided into two parts. The first part presents the basis of two approaches for problems solving: Constraint Programming and Interval Analysis. The second part contains the contributions of the thesis.

The first part is composed of three chapters. In Chapter 2, an introduction to Constraint Programming (centered on finite domains) and a set of techniques for problem solving are given. Chapter 3 introduces Interval Analysis and the set of tools to obtain reliable results in computational operations. These tools include techniques for solving systems of non-linear equations in a numerical way. Chapter 4 presents the combination of Constraint Programming and Interval Analysis as complementary approaches for problem solving.

The second part is composed of four chapters. Chapter 5 discusses a first approach for solving a system of distance equations with uncertainties and some limitations of it. A new strategy based on conditional bisection is then introduced in order to overcome some of these liminations. In Chapter 6, an application of this and other techniques for a special class of parallel robot is discussed. Chapter 7 introduces two techniques for detecting boxes included inside the solution set of a problem. These techniques are called *inner boxes tests*. Chapter 8 presents a new technique for extending domains of the variables in a consistent way (ensuring that all values are solutions of the problem). Finally, Chapter 9 contains the conclusion of this thesis.

Part I

State of Art and Related works

Chapter 2

Constraint Programming

A basic principle of evolution is adaptation. Most species must find a compatible state with the set of constraints imposed by their environment. It may seem that it is not a frequent problem (or that it is restricted only to big changes in the nature), but actually we always face the problem of finding a compatible state with the set of constraints imposed by our environment. By *environment* we can understand our family, our work or activities, the city in which we are living, etc. Everything that we cannot control is a part of our environment.

Consider, for example, a man in a supermarket trying to put his objects in a little box. He has essentially a space constraint and must find a good arrangement for all the objects in the box. A woman needs to buy several things in different parts of a big city during the day must find a good schedule for her visits. She has a time constraint. A boy painting a map with a limited number of colors, must be sure that two adjacent countries have not the same color. He has a compatibility constraint.

In all these examples we have some things that we can control and some constraints that we must respect. This is exactly the cornerstone of an important and powerful paradigm in computer science called *Constraint Programming*.

Constraint Programming (CP) is a declarative programming technique that has grown from the collaboration of several research communities including Artificial Intelligence, Computational Logic, Programming Language, Mathematics, and Operations Research. It is a good framework for modeling and solving difficult combinatorial and optimization problems. A problem is typically characterized as a state of the world containing a number of objects, and a set of relationships between them that must be verified.

In this framework, a generic form to represent a problem is through a *Constraint Satisfaction Problem* (CSP), that is a model built from a set of variables with associated domains and a set of relationships between these variables, called constraints. A *solution* of the problem is an instantiation for each variable such that all constraints are satisfied. This solution is commonly obtained with a *solving* process that combines *search* and *filtering* techniques.

The aim of this chapter is to give a brief introduction of these concepts, that will be useful thereafter. Section 2.1 presents a short introduction of Constraint Programming and the main contributions received from other communities. Section 2.2 introduces some concepts like CSP, equivalent CSP, CSOP, among some others in a more formal way. Section 2.3 presents the main tools for solving Constraint Satisfaction Problems, including complete algorithms, consistency techniques and combination of them. Finally, section 2.4 presents the conclusions of this chapter.

2.1 A Short Introduction

The Constraint Programming community was formally born at the beginnings of the nineties. Is is based on ideas coming from different communities as Artificial Intelligence, Operations Research, Logic Programming and Mathematics. One of its main characteristics is the use of *constraints* for representing a problem. Thus, *solving the problem* means to find an instantiation of a set of variables that verifies these constraints (see [135]).

In this framework, the process of solving a real-life problem can be separated in two parts: the model, and the solving phase.

The model part means to express the real-life problem in the form of a set of variables, domains and constraints; which is called a Constraint Satisfaction Problem (CSP). This idea was originally introduced by the Artificial Intelligence community (see [136, 150]) and later used in other communities like Constraint Logic Programming and Operations Research.

The solving phase combines different strategies for searching solutions in the space given for the domains of the variables. This search can be performed systematically or stochastically, and can combine several tools as the *consistency techniques* (introduced in the area of Artificial Intelligence [136, 150, 208]), *tree search* and *backtracking* techniques (commonly used in the area of Logic Programming [38, 39, 40]), *simplex method* (used in Operations Research for problems

with linear constraints over reals domains [47]), and combinations of those (see [91, 157]).

In order to perform this process automatically, many of these strategies have been implemented in specific solvers or programming languages. One of the first implementations of constraint programming languages was CHIP (Constraint Handling In Prolog), introduced by Pascal van Hentenryck [93] and based on PROLOG. It contains some ideas introduced by Jean-Louis Laurière [129], who proposed the ALICE system (a general system for problem solving using logic).

CHIP introduced in logic programming new domains of computation: the integer, rational, and boolean domains. It also combined the approaches of logic programming with combinatorial search and constraint propagation in a new paradigm characterized by its declarative nature [63, 106].

As CHIP, many other implementations of constraint programming have been proposed in the form of libraries for particular programming languages or specific solvers. Some examples are ILOG Solver [104, 105], Mozart-OZ [192], ECLiPSe [5], Choco (Claire) [26], CHR [62], and JSolver [32]. A comparative study of some of them can be found in [58].

2.2 Constraint Satisfaction Problems

The Constraint Satisfaction Problems (CSPs) have a long history from the works of Waltz [208], Montanari [150], and Mackworth [136]. A CSP is basically a *Model* which represents a real problem using a set of decision objects called variables.

From a formal point of view, a CSP is defined as follows:

Definition 1 (CSP) *A CSP is a triple* (X, D, C) *where* $X = \{x_1, \ldots, x_n\}$ *denotes a set of variables,* $D = \{D_{x_1}, \ldots, D_{x_n}\}$ *denotes a set of associated domains and* $C = \{c_1, \ldots, c_m\}$ *denotes a set of constraints.*

Example 2.2.1 *Consider the problem of the boy painting a map with a limited number of colors (presented at the beginning of this chapter), and consider a set of variables* $x_1, \ldots, x_n$ *representing each country in the map. Let* $D = \{blue, red, yellow, ...\}$ *be the set of available colors for painting a country. This problem can be easily modeled with the following CSP:*

$P = (X, D, C)$
$X = \{x_1, \ldots, x_n\}$

$$D = \{D_{x_i} = \{blue, red, yellow, ...\} \quad \forall i = 1, \ldots, n\}$$
$$C = \{c_{ij} : x_i \neq x_j, \quad \forall i \text{ adjacent to } j\}$$

Notice the declarative nature of the CSP. This model only gives the set of characteristics that a solution must verify. Notice also that domain values do not need to be a set of integers (although often they are), nor numerical. Some examples of domains are:

- Boolean: in which only the values **true** or **false** are considered.
- Finite: in which a finite set of values or symbols are considered.
- Continuous: in which values are represented by intervals.
- Mixed: involving two or more of the above.

The set of all possible combinations of values from the domains of the variables is called the *search space*, and is defined as follows:

Definition 2 (Search Space) *Let $P = (X, D, C)$ be a CSP with domains $D = \{D_{x_1}, \ldots, D_{x_n}\}$. The sub-space $E = D_{x_1} \times \ldots \times D_{x_n}$ is called the* search space *associated to P and corresponds to the Cartesian product of the domains of the variables.*

An element of the search space that verifies all the constraints is called a *solution*. The set S of all solutions of the CSP in the search space is called *the solution space*.

Some CSPs also require a solution that maximizes (or minimizes) an *objective function*[1]. They are commonly called *Constraint Satisfaction and Optimization Problems* (CSOPs). As an example, consider the same problem presented in example 2.2.1, but with the following additional condition: *minimize the number of colors used for painting the map*. Clearly, it becomes an optimization problem and therefore a CSOP.

A real-life problem can often be represented in different ways, and therefore it is possible to associate different CSPs. Actually, two CSPs are said *equivalent* if they are the same solution space.

[1] Also called *cost function* in the Operations Research literature.

2.2.1 Constraints

The constraints are relations that restrict the possible values of the variables. A constraint affecting only one variable is called *unary constraint*, while a constraint affecting two variables is called *binary constraint*. Obviously a constraint affecting n variables is called *n-ary constraint*, but only unary and binary constraints will be considered in this section.

If all constraints of a CSP are at most binary constraints, the CSP can be easily represented using a *constraint graph*. In a constraint graph, the variables are represented by nodes and the constraints by edges. For example, consider the problem of painting the map of the South American continent presented in Figure 2.1, using four colors: blue, yellow, red, and green. This problem can be modeled

Figure 2.1: Map of the South American continent.

with the following CSP:

$$P = (X, D, C)$$
$$X = \{Ch, Ar, Ur, Br, Pa, Bo, Pe, Ec, Co, Ve, Gu, Su, GF\}$$

$D = \{D_k = \{blue, yellow, red, green\} \quad \forall k = Ch, \ldots, GF\}$

$C = \{Ch \neq Ar,\ Ch \neq Bo,\ Ch \neq Pe,\ Ar \neq Ur,\ Ar \neq Br,\ Ar \neq Pa,\ Ar \neq Bo, Ur \neq Br, Br \neq Pa, Br \neq Bo, Br \neq Pe, Br \neq Co, Br \neq Ve, Br \neq Gu, Br \neq Su, Br \neq GF, Pa \neq Bo, Bo \neq Pe, Pe \neq Ec, Pe \neq Co, Ec \neq Co, Co \neq Ve, Ve \neq Gu, Gu \neq Su, Su \neq GF\}$

and represented with the constraint graph shown in Figure 2.2.

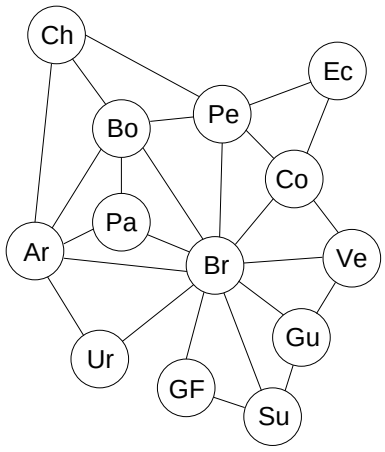


Figure 2.2: Constraint graph of the problem.

2.3 Solving Phase

The process of solving a CSP may have different meanings. For example:

- to determine whether a solution exists,
- to find one or all solutions, or
- to find an optimal solution relative to an objective function.

The strategy used for performing this process depends on the type of the domains of the variables. In this chapter, only CSP over finite domains will be considered, because the most of the initial works in Constraint Programming are related to these types of domains. CSPs over continuous domains and their relations with Interval Analysis will be presented in Chapter 4.

When finite domains are considered, the search space of a CSP is composed by a finite number of elements that increases exponentially in the number of variables. This phenomenon is known as *combinatorial explosion*.

For some types of problems, there exists an algorithm that is able to solve them in a polynomial time (in the worst case), but for the general CSPs this algorithm does not exist.

The problems for which finding an answer takes a *nondeterministic polynomial time* are called NP. For this type of problems it is possible to quickly (in polynomial time) test whether a solution is correct, but finding this solution is generally a hard task.

Among the NP problems, there exists a special family of problems known as NP-complete. In this family, any problem can be *reduced* to another one (that means, it is possible to transform a problem into a particular case of another one using a polynomial time worst case algorithm). Therefore, if one problem of this family could be solved in polynomial time, every problem in the family could be solved in polynomial time. Anyway, many studies suggest that it is far to be possible.

In the case of CSPs with finite domains (presented in this chapter), knowing if there exists a solution to them (in the general case) is NP-complete. For this reason, most of the works in Constraint Programming try to improve the performance of the searching process, even though the process keeps non polynomial in the worst case.

2.3.1 Searching Process

In order to find solutions in the search space of a CSP over finite domains, a searching process must be performed. This process can be performed by searching systematically through the possible assignments of values to the variables, or by searching stochastically elements of the search space. Techniques based on the first idea are called *complete techniques*, while the techniques based on stochastic search are called *incomplete*.

Complete algorithms guarantee to find a solution if one exists or prove that no solution exists. They are based on tree search and combine domain reduction and constraint propagation in order to eliminate values from the domain of the variables, which cannot be part of any solution of the problem. Their main drawback is that, in the worst case, these algorithms need to examine all the elements of the search space and this process is sometimes computationally unfeasible for real-life problems.

Incomplete algorithms generally use local search methods [163, 1], and address the problems that cannot be solved by complete algorithms. Local search is based

on the concept of a neighborhood, which means to look for a solution near to a given initial candidate solution. Among the most popular incomplete techniques are Hill Climbing [177], Taboo search [69], and Simulated Annealing [117]. Genetic and Evolutionary Algorithms [146] are also popular incomplete techniques, but they are based on the simultaneous evolution of a set of candidate solutions and generally address optimization problems.

In this document, only complete techniques are considered, mainly because they give a guaranteed response. So, the expression *solving the problem* means to find all the solutions of the problem (or to prove that no solution exists).

2.3.2 Complete Algorithms

Complete algorithms are based on the idea of *searching systematically* in the whole search space of a problem. It means to test the elements of the search space and to identify those that verify all constraints. The simplest method to perform this task is called *Generate-and-test*. The main idea is to generate all possible combinations of the values from the domains of the variables and to test if the generated element is feasible (the element verifies every constraint). Its main drawback is the number of generated candidates (that is equal to the cardinal of Cartesian product of all the variable domains).

A better method is to use a tree search algorithm. These algorithms were first introduced in the Artificial Intelligence community. The three basic algorithms are [177]: *Depth-first search*, *Breadth-first search*, and *Best-first search*.

In Logic Programming, the Depth-first search algorithm is implemented under the name of *Backtracking* algorithm and combined with the concept of *failure in the instantiation*. In backtracking, the main idea is to instantiate the variables sequentially, and to check a constraint as soon as the associated variables are instantiated. If a partial instantiation violates any of the checked constraints, then the value of the last instantiated variable is changed and the constraints are re-checked. If no more values are available in the domain of a variable v, a failure is detected and the algorithm comes back to the variable instantiated before v. This is the backtracking process, and it allows one to eliminate a part of the search space (that cannot contains any solution), improving the performance of the search.

Anyway, if all the constraints of the problem involve all variables, backtracking and generate-and-test will generate all possible combinations of the values in the search space.

From now on, only binary CSPs will be used to explain the techniques associated to the solving phase on problems over finite domains. On one hand due to their simplicity, and on the other hand because any CSP involving n-ary constraints can be transformed into an equivalent CSP involving at most binary constraints (as it has been shown in [180]).

Example 2.3.1 *Consider the following CSP:*

$$P = (X, D, C)$$
$$X = \{x, y, z\}$$
$$D = \{D_x = \{1,2,3,4\}, \quad D_y = \{1,2,3,4\}, \quad D_z = \{1,2,3\}\}$$
$$C = \{c_1 : x < y, \quad c_2 : y \geq z, \quad c_3 : z > 2\}$$

Table 2.1 presents the results of applying both the generate-and-test and the backtracking algorithms for solving this CSP.

Generate-and-test x, y, z			**Backtracking** x, y, z		
(1, 1, 1)	(2, 2, 2)	(3, 3, 3)	(1, _, _)	(2, 2, _)	(3, 4, 3)
(1, 1, 2)	(2, 2, 3)	(3, 4, 1)	(1, 1, _)	(2, 3, _)	(4, _, _)
(1, 1, 3)	(2, 3, 1)	(3, 4, 2)	(1, 2, _)	(2, 3, 1)	(4, 1, _)
(1, 2, 1)	(2, 3, 2)	(3, 4, 3)	(1, 2, 1)	(2, 3, 2)	(4, 2, _)
(1, 2, 2)	(2, 3, 3)	(4, 1, 1)	(1, 2, 2)	(2, 3, 3)	(4, 3, _)
(1, 2, 3)	(2, 4, 1)	(4, 1, 2)	(1, 2, 3)	(2, 4, _)	(4, 4, _)
(1, 3, 1)	(2, 4, 2)	(4, 1, 3)	(1, 3, _)	(2, 4, 1)	
(1, 3, 2)	(2, 4, 3)	(4, 2, 1)	(1, 3, 1)	(2, 4, 2)	
(1, 3, 3)	(3, 1, 1)	(4, 2, 2)	(1, 3, 2)	(2, 4, 3)	
(1, 4, 1)	(3, 1, 2)	(4, 2, 3)	(1, 3, 3)	(3, _, _)	
(1, 4, 2)	(3, 1, 3)	(4, 3, 1)	(1, 4, _)	(3, 1, _)	
(1, 4, 3)	(3, 2, 1)	(4, 3, 2)	(1, 4, 1)	(3, 2, _)	
(2, 1, 1)	(3, 2, 2)	(4, 3, 3)	(1, 4, 2)	(3, 3, _)	
(2, 1, 2)	(3, 2, 3)	(4, 4, 1)	(1, 4, 3)	(3, 4, _)	
(2, 1, 3)	(3, 3, 1)	(4, 4, 2)	(2, _, _)	(3, 4, 1)	
(2, 2, 1)	(3, 3, 2)	(4, 4, 3)	(2, 1, _)	(3, 4, 2)	

Table 2.1: Results using Generate-and-test and Backtracking algorithms.

In the example 2.3.1, the problem has only five solutions (which are underlined in the table 2.1). Both algorithms obtain the same solutions, but backtracking generates only 38 elements instead of 48. It is because backtracking tests a partial instantiation as soon as possible and eliminates a part of the search space when a partial instantiation does not verify all the constraints.

Notice that the set of generated elements depends on the order in which variables are instantiated. For this reason, different heuristics for selecting the next variable to be instantiated have been proposed. Among the most popular heuristics are:

- selecting the next variable randomly,
- the variable with the smallest domain (also called the *fail-first*),
- the variable involved in the most of the constraints (also called *maximum degree*).

These heuristics have shown to effectively reduce the number of generated elements and therefore to improve the computing time in the backtracking algorithm.

Table 2.2 shows the results of applying the same backtracking algorithm with different instantiation orders.

Backtracking z, x, y			**Backtracking** z, y, x	
(_, _, 1)	(2, _, 3)	(3, 3, 3)	(_, _, 1)	(3, 3, 3)
(_, _, 2)	(2, 1, 3)	(3, 4, 3)	(_, _, 2)	(4, 3, 3)
(_, _, 3)	(2, 2, 3)	(4, _, 3)	(_, _, 3)	(_,4, 3)
(1, _, 3)	(2, 3, 3)	(4, 1, 3)	(_,1, 3)	(1, 4, 3)
(1, 1, 3)	(2, 4, 3)	(4, 2, 3)	(_,2, 3)	(2, 4, 3)
(1, 2, 3)	(3, _, 3)	(4, 3, 3)	(_,3, 3)	(3, 4, 3)
(1, 3, 3)	(3, 1, 3)	(4, 4, 3)	(1, 3, 3)	(4, 4, 3)
(1, 4, 3)	(3, 2, 3)		(2, 3, 3)	

Table 2.2: Results using Backtracking with some heuristics.

The backtracking z, x, y first instantiates the variable with the fewest values in the domain (because a failure of this instantiation will perform a big pruning in the search space). If two variables have the same number of values in their domains, then the alphabetic order is used.

The backtracking z, y, x uses both the fail-first and the maximum degree heuristics. It first instantiates the variable with the fewest values in the domain, but if two variables are the same number of values, then the maximum degree heuristic is used (choosing the variable that is involved in the largest number of constraints). This backtracking algorithm reduces the search space by early pruning zones that cannot contain solutions due to an incompatible value in the domain of a variable, and generates only 15 elements of the search space.

Sometimes, only one solution to the problem is needed. In this case, the number of generated elements until finding a solution, depends on the order in which values from the domains of the variables are selected. Among the well-known value selection heuristics are random selection, minimum value selection, and the value with the greatest probability of success (in this case additional information about the problem is needed).

Although these heuristics improve the performance of backtracking, there exists another important drawback called the *thrashing problem* [67]. It is considered as the main drawback of the pure backtracking algorithm and consists in searching in different parts of the search space failing for the same reason.

In the results of Table 2.2, the first three values of the backtracking are due to the constraint $c_3 : z > 2$. This column would have a lot of values if the domain of the variable z were bigger (e.g. $\{-100, -99, \ldots, 3\}$). The same effect appears in the backtracking column of Table 2.1 each time that variable z takes a value less than 3.

This situation can be avoided by using some techniques to eliminate inconsistent values from the domain of the variables, and therefore to prune parts of the search space. These techniques are called *consistency techniques* and perform two important processes in Constraint Programming: *domain reduction* and *propagation*. Consistency techniques are deterministic (as opposed to the search which is non-deterministic), and sometimes they can find the solution of a problem without performing any search phase. Nevertheless, they are rarely used alone.

The combination of backtracking, consistency techniques and heuristics for choosing the next variable to be instantiated or the next value to be assigned are the basis of several strategies for searching solutions. Some of them use also the constraint graph for detecting the reason of a failure.

Among these strategies are the *look-back strategies* (e.g. backjumping and intelligent backtracking), and the *look-ahead strategies* (e.g. forward checking).

Look back strategies try to determine the actual reason of a failure in a partial instantiation. The main idea is to change the value of the variable that caused the failure instead of the last instantiated variable, and therefore to avoid the thrashing problem.

Look-ahead strategies try to determine if a current partial instantiation is compatible with the set of uninstantiated variables by applying a consistency technique. If after this process one of the variable domains becomes empty, the last instantiation can be considered as not possible, and the backtracking process

is immediately performed.

More information about these and other strategies can be found in [91, 157] and [51, 194]. A good survey on algorithms for Constraint Programming can be found in [128]. In the next section, a brief introduction of consistency techniques will be presented. In Section 2.3.4, these techniques will be combined with backtracking in order to show the advantages of a Look-ahead approach.

2.3.3 Consistency Techniques

Consistency techniques were first introduced for improving the efficiency of picture recognition programs by researchers in artificial intelligence (see [208, 150]). In his work, Waltz [208] was interested in the problem of automatically generating three-dimensional representations from line drawings of scenes. Roughly speaking, his approach involved labelling all the lines in a picture in a consistent way. As the number of possible combinations can be huge, he introduced a sort of *filtering program* to systematically remove all inconsistent labellings at a very early stage, and thus reduce the number of combinations in a tree search.

The work of Montanari [150] introduced the idea of *networks of constraints*, and the possibility of using these networks for removing inconsistent values from the domain of the *variables*. Finally, Mackworth [136] integrated the previous works in his paper *Consistency in networks of relations*, in which he presented, in a general framework, some consistency algorithms like *node*, *arc* and *path* consistency.

The main idea behind a consistency technique is to eliminate from the domain of the variables all the values that cannot belong to a solution of the problem. This section discusses three of the most popular consistency properties and some algorithms to achieve it. For simplicity, the explanation will be based on the following *binary* CSP:

$$
\begin{aligned}
&P = (X, D, C)\\
&\quad X = \{x, y, z, w\}\\
&\quad D = \{\{1,2,3\}, \{2,3,4\}, \{2,3\}, \{2,3\}\}\\
&\quad C = \{c_1 : x > 1; c_2 : x = y; c_3 : y \neq z; c_4 : z \neq w; c_5 : y \neq w\}
\end{aligned}
$$

The constraint graph of the CSP is shown in figure 2.3.

Node Consistency

This is the simplest consistency technique and involves the unary constraints in the constraint graph.

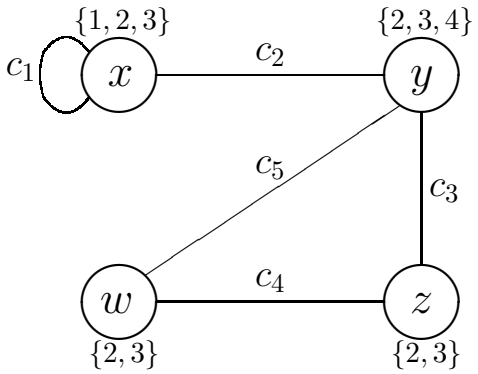


Figure 2.3: Constraint graph of the problem.

Definition 3 (node-consistency) *Let N a node representing a variable x in the constraint graph, and let D_x be the domain of the variable x. N is node consistent iff for any value $v \in D_x$ each unary constraint on N is satisfied.*

This property is based on the following observation: *if a value v in the domain of a variable x does not satisfy a unary constraint, then the instantiation of x to v will always result in immediate failure.*

In the example of figure 2.3, node x is not node-consistent because the value $x = 1$ does not verify the constraint $c_1 : x > 1$. The node inconsistency can be eliminated by removing those values that do not satisfy unary constraints. This process is called **domain reduction**, and has a considerable impact in a later application of the backtracking algorithm. Notice that deleting such values does not eliminate any solution of the original CSP (it is a basic property of any consistency technique). The procedure for achieving node consistency is presented in algorithm 1.

Algorithm 1 Node consistency

```
for all N in Nodes(G) do
  for all v in the domain D of N do
    if any unary constraint on N is inconsistent with v then
      delete v from D
return
```

When all unary constraints are verified, and there is no value in the domain of the variables that violates the unary constraints, the CSP is called node-consistent. Figure 2.4 shows the final equivalent CSP (which is node-consistent).

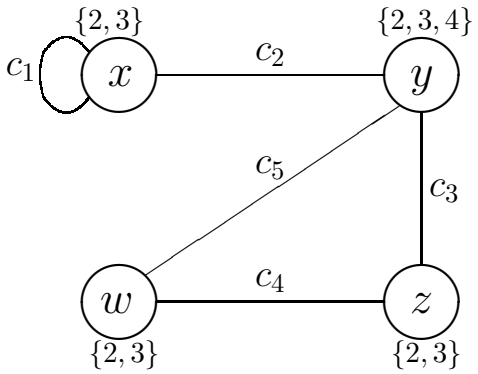


Figure 2.4: Result of applying node-consistency over the CSP.

Arc Consistency

This is one of the most popular consistency techniques and has been an inspiration for other classes of consistency techniques used, for example, in continuous domains (see section 4.3.1). It is based on the notion of *support* of a value in the domain of the other variables. A value v of the domain of a variable x has a support in the domain of another variable y iff there exists at least one value in the domain of y, such that the constraints between them are satisfied.

Definition 4 (arc-consistency) *An arc $Arc(x, y)$ between two variables x and y is arc consistent iff for every value $v \in D_x$, there is some value $w \in D_y$ such that $x = v$ and $y = w$ is permitted by the binary constraints between x and y.*

Notice that this property is *directional*, i.e., if an arc $Arc(x, y)$ is consistent, then it does not automatically mean that $Arc(y, x)$ is also consistent. For example, consider the node x with domain $D_x = \{2, 3\}$ (node consistent), and the node y with domain $D_y = \{2, 3, 4\}$ of the figure 2.4. The arc $Arc(x, y)$ is arc consistent, while the arc $Arc(y, x)$ is not (because the value $y = 4$ does not have any *support* in the domain of the variable x for the constraint $x = y$).

As in the case of node consistency, an arc $Arc(x, y)$ can be made consistent by simply deleting those values from the domain of x for which there does not exist any support in the domain of y (observation made by Fikes [59] for discrete domains).

A CSP is arc-consistent if all its arcs are consistent. Algorithm 3 presents the procedure AC-3 for achieving arc-consistency. This algorithm is based on the procedure REVISE(x,y) shown in Algorithm 2. Notice that after applying

the procedure REVISE to an arc $Arc(x, y)$, this arc becomes consistent; however, some changes in the domain of the variable x can produce inconsistent values in the other arcs of the graph. The procedure AC-3 includes in a queue these possible inconsistent arcs.

Algorithm 2 REVISE(x,y)

```
DELETE ← false
for all v ∈ D_x do
  if there is no w ∈ D_y such that Arc(x, y) is consistent then
    delete v from D_x
    DELETE ← true
return DELETE
```

Notice also that AC-3 algorithm performs a revision only for those arcs that are possibly affected by a previous revision. Moreover, after applying REVISE(x,y), it is not necessary to add arc (y, x) to the queue because none of the elements deleted from x provides a support for any value in y.

Here is the other important process of Constraint Programming, which is called **constraint propagation**. When the domain of a variable is changed (by deleting one or more of its values), the change is *propagated* across the whole network. Every variable which is possibly affected by this change is again revised, and every value which becomes inconsistent is removed. The process continues until the whole network becomes consistent.

Algorithm 3 Arc-consistency (AC-3)

```
Q ← {(x, y)|(x, y) ∈ arcs(G), x ≠ y}
while Q not empty do
  select and delete any arc (x, y) from Q
  if REVISE(x,y) then
    Q ← Q ∪{(z, x)|(z, x) ∈ arcs(G), z ≠ x, z ≠ y}
return
```

After applying AC-3 to a CSP, it becomes arc-consistent, but it does not mean that the problem has a solution. Consider, for example, the resulting arc-consistent CSP shown in Figure 2.5.

Even though this CSP is arc-consistent, it does not have any solution[2].

[2]It is because arc-consistency and node-consistency only use the information of one constraint at once. Other techniques like *path consistency* try to avoid this limitation by considering more than one constraint for detecting inconsistent values.

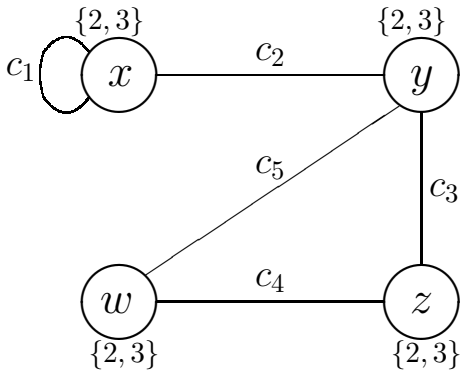


Figure 2.5: Results of applying arc-consistency over the CSP.

There exists only one case in which the arc-consistency technique gets the solution of the problem (without using a backtracking or similar process after), and that is when the domain size of each variable becomes one. In this case, the CSP has a single solution, and can be obtained by assigning to each variable the only possible value in its domain.

Path Consistency

Path consistency technique (introduced by Montanari [150]) is a generalization of the arc-consistency technique. Path consistency tries to improve the detection of inconsistent values in the domain of the variables by considering more than one constraint at the same time.

A CSP will be path-consistent if for every pair of values v_x, v_y allowed by a constraint between two variables x and y, it is also allowed by all paths from x to y. In other words, the CSP will be path-consistent if there exists at least a compatible intermediate vertex value that satisfies the unary and binary constraints from x to y.

Montanari has shown that a CSP is path-consistent if all paths of length 2 are path consistent. Therefore, an algorithm to enforce path-consistency needs only triples of variables to work (path of length 2).

Consider again the CSP shown in Figure 2.5, which is arc-consistent. The value 2 in the domain of the variable w has a compatible value 3 in the domain of y, but has no value v_z in the domain of the variable z such that $(2, v_z)$ and $(v_z, 3)$ are allowed simultaneously. Therefore, the pair $(2, 3)$ is not allowed for the constraint c_5 between w and y. The same occurs with the pair $(3, 2)$. As the

pairs $(3, 3)$ and $(3, 3)$ the application of a path-consistency algorithm removes all possible combination of values between y and w, and then proves that no solution is possible for this CSP.

Singleton Consistency

The main idea behind any consistency technique is to detect inconsistent values in the domain of the variables in order to remove them. Consider a CSP $P = (X, D, C)$ and a variable $x \in X$. If there exists a solution for P in which the variable x can take the value v, then none consistency technique applied to P when $D_x = \{v\}$ can produce an empty domain. In other words, if a consistency technique applied to P with $D_x = \{v\}$ produces an empty domain in a variable, then the value v is not consistent (and cannot belong to any solution).

This is the basic idea in a singleton consistency technique. For example, singleton arc-consistency [49] deletes the value v from the domain of a variable x if a CSP $P = (X, D, C)$ cannot be arc-consistent when $D_x = \{v\}$. Actually, any consistency technique can be applied to detect the possible inconsistency of a CSP when one of its variables takes a specific value. Moreover, if a local consistency can be enforced in a polynomial time, the corresponding singleton consistency can also be enforced in a polynomial time (see [168] for some examples).

Bound Consistency

In problems involving ordered numeric domains, another technique for detecting inconsistencies is called bound-consistency [169]. The main idea is to check only the bounds of the ordered domains instead of checking all values in the domain.

Definition 5 (bound-consistency) *An n-ary constraint c involving the variables* $(x_1, \ldots, x_n)$ *is called bound-consistent if and only if for each variable* x_i : $\forall d_i \in \{\min D_i, \max D_i\}, \forall j \in \{1, \ldots, n\} - \{i\}, \exists d_j \in [\min D_j, \max D_j]$ *such that* $(d_1, \ldots, d_m)$ *verifies c.*

The main advantage of this consistency technique is that only the bounds of the domains are verified. Therefore, less time is needed to verify a bound consistent constraint. Moreover, this consistency can be applied with special domains (e.g. continuous domains), for which classic arc-consistency techniques are unfeasible.

Generalized Arc-consistency

The original arc-consistency technique has a sense only in binary CSPs because binary CSPs can be represented as a graph. In the case of general non-binary constraints, there exists an equivalent arc-consistency technique called *generalized arc-consistency* (GAC) (see [20]). A constraint is GAC if for any value of the variable in the constraint there exist values for the other variables in the constraint such that the constraint is satisfied.

Improvement of Consistency Techniques

Many improved algorithms for achieving arc-consistency and path consistency have been proposed. In the case of arc-consistency, an algorithm having an optimal worst-case time complexity (known as AC-4) was introduced in [149]. As the AC-4 algorithm has still a not optimal space complexity, a new arc-consistency algorithm with optimal time and space complexity was introduced in [17]. This algorithm is known as AC-6. Other examples of improvement can be found in [86, 31], specially in the path-consistency algorithm.

In [60, 42], a generalization of the consistency techniques is introduced under the concept of $k - consistency$, showing that node, arc and path consistency can be considered as particular cases of 1-consistency, 2-consistency and 3-consistency, respectively.

Some stronger consistency techniques (k-consistency with $k > 3$) have been proposed, but they are in general too expensive to be applied in practice.

Combinations of consistency techniques and heuristics have introduced many other improvement in the so-called *filtering* algorithm. Some examples can be found in [50, 173, 213, 200]).

The development of specific consistency techniques for global constraints is another of the most important improvements in the area of Constraint Programming. Specific algorithm for the *AllDiff* constraints [174, 201] or the *Cardinality* constraints are only some examples.

2.3.4 Backtracking and Consistency Techniques

A basic idea to improve the performance of the search process is to perform a consistency technique before starting a backtracking algorithm. Thus, each value of the domain of the variables is checked in a early phase, and if some inconsistency is detected the inconsistent value is deleted immediately.

A better idea is to check for inconsistencies at each step of the search because the instantiation of a variable with a given value can produce an inconsistent situation in the remaining variables. There are different Look-ahead algorithms that perform this process. One of the most popular is the Forward Checking algorithm [91].

Forward Checking

The main idea of the Forward Checking algorithm is to (temporarily) remove from the domain of the not yet instantiated variables the set of values that are incompatible with the last instantiated variable. Only variables related to the last instantiated variable are considered. If the domain of one or more of these variables becomes empty, then the (previously) deleted values of the domains of the checked variables are restored and the current value of the last instantiated variable is changed. If no value is consistent then the backtrack process is performed.

Forward Checking guarantees that at each step the current partial solution is consistent with each value in each not yet instantiated variable.

Example 2.3.2 *Consider the following CSP:*

$P = (X, D, C)$

$X = \{v, w, x, y, z\}$

$D = \{D_v = \{2,5,6\}, D_w = \{2,4,5,7\}, D_x = \{2,4\}, D_y = \{2,4,5\}, D_z = \{1,2,3,4,5\}\}$

$C = \{c_1 : v \geq w, c_2 : w \neq x, c_3 : w \neq y, c_4 : x \neq y, c_5 : y = z, c_6 : v \geq z\}$

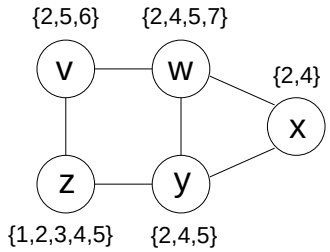


Figure 2.6: Graph of constraints with initial domains.

This CSP is represented in Figure 2.6 in the form of a graph of constraints. After applying an arc-consistency technique the search space of the problem is reduced from 360 elements to 108, as shown in Figure 2.7(a).

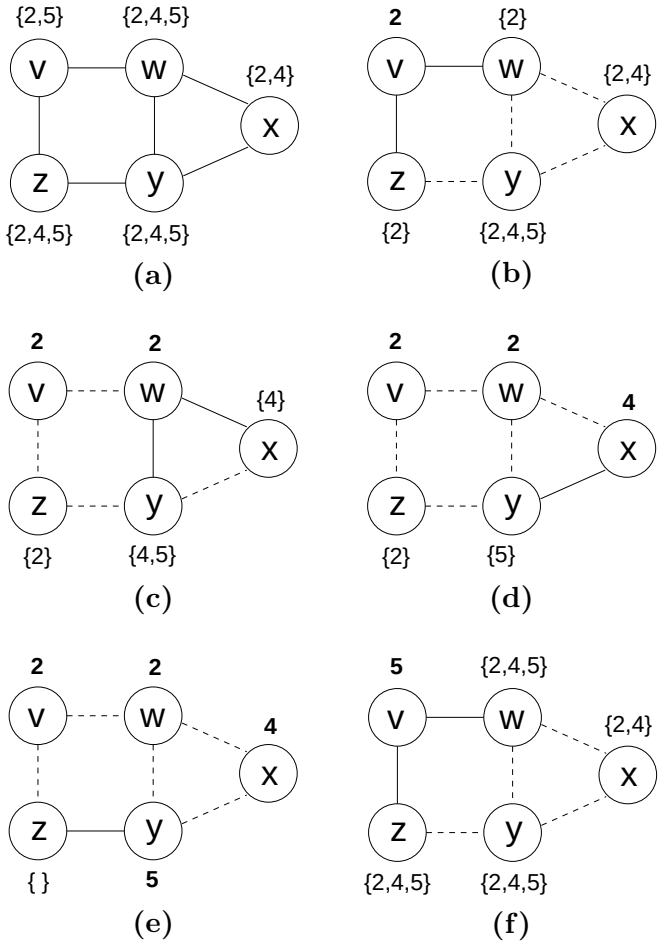


Figure 2.7: (a) CSP arc-consistent. (b) Instantiation of the variable v and forward check of the variables w and z. (c) Instantiation of w and forward check of x and y. (d) Instantiation of x and forward check of y. (e) Instantiation of y and detection of the inconsistency: the domain of z is empty. (f) The backtracking process returns until the variable v (restoring the deleted values in each step) and the variable v is instantiated with the next value.

The search process starts with the instantiation $v = 2$ and the forward check of the related variables w and z. This process continues with the instantiation of w, x, and y respectively. After the instantiation of the variable y, the domain of z becomes empty, and a consistency fail is detected. At this point, the deleted value from the domain of the variable z is restored. The value of the last instantiated variable must be changed, but y has no another value in its domain. The backtrack process is then preformed, restoring the deleted values from the domain of y and

going back to the variable x. The backtrack process will return until the variable v, deleting the value 2 from its domain and performing a new instantiation with the next value 5.

This technique has been generalized by van Hentenryck [93] in order to handle n-ary constraints. The main idea is to performed the forward check over an uninstantiated variable as soon as the n-1 variables of an n-ary constraint have been instantiated. In such cases the n-ary constraint is called *forward checkable*. A stronger generalization of forward checking to n-ary constraints has also been proposed in [18].

Full Look Ahead

Although Forward Checking effectively improves the performance of the searching process, it still suffers from late detection of inconsistencies. For example, in Figure 2.7, the insconsistent instantiation $v = 2$ is detected only after the instantiation of the variable y.

A better approach for early detection of inconsistencies was introduced in [183] as Maintaining Arc Consistency (MAC). It is a full look ahead technique that enforces a complete consistency technique after each instantiation of a variable. No only the related variables are checked but all the remaining uninstantiated ones. This algorithm guarantees that at each step the current partial solution has an arc-consistent sub-network.

Figure 2.8 shows the MAC technique applied to the same problem of example 2.3.2. The process starts with the application of arc-consistency to the whole system. The first instantiation of the variable v yields to an inconsistent sub-network (detected by arc-consistency). This value is deleted, and the domains restored. A new instantiation is performed ($v = 5$), and a consistent network is detected. The instantiation of the next variable ($w = 2$) produces an arc-consistent network with only one value in the domain of the remaining uninstantiated variables. Therefore, a solution is immediately detected.

In [19], a comparison between Forward Checking and MAC has been reported. The results suggest that MAC is more efficient than Forward Checking to solve not only large practical problems but also hard and large random problems. A combination of these techniques with different variable ordering heuristics is also presented.

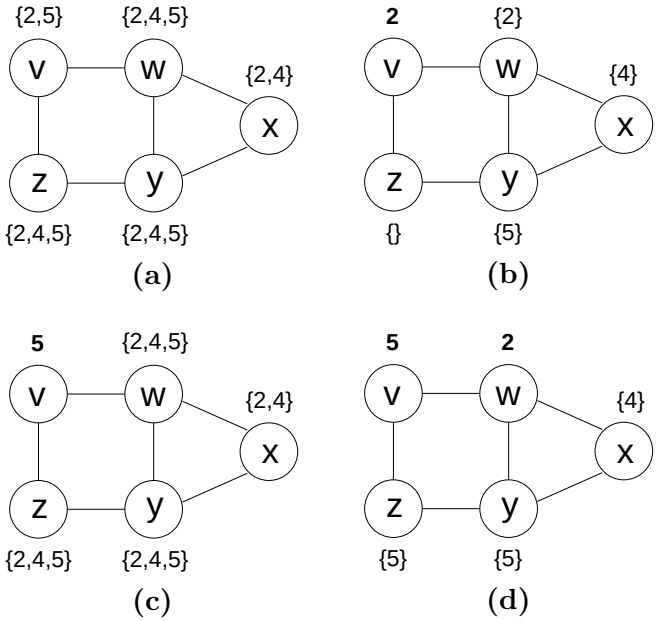


Figure 2.8: (a) CSP arc-consistent. (b) Instantiation of v with the value 2 and detection of inconsistency. (c) Instantiation v with the value 5 and the remaining arc-consistency network. (d) Instantiation of w with the value 2 and detection of the first solution.

2.4 Conclusions

Constraint Programming has been one of the important advances in programming since the last two decades. As a framework, it allows one to model and solve difficult combinatorial and optimization problems in an efficient way.

In this chapter, a brief introduction to Constraint Programming with finite domains and its associated tools has been presented. Important concepts about constraint programming, CSPs solving techniques, complete and incomplete search algorithms, and heuristics for variable and value ordering have been presented. Some important elements to keep in mind are the formal model of a real-life problem (the CSP) and the solving process which involves the concepts of *search space*, *solution* and *solution space*, *domain reduction*, and *constraint propagation*.

These ideas, combined with others tools from Interval Analysis community provide a good framework for solving difficult non linear problems over continuous domains, and they are the bases of the works developed in this thesis.

Chapter 3

Interval Analysis

Computers work with digital numbers, not with real ones. This is a big difference between a *theoretical result* and a *practical result*. In order to express a real number, a computer will usually truncate and round it. Therefore, the *digital number* computed (or floating-point number [70]) is only an approximation of the original number. This simple fact produces important differences in computed results when many operations are performed, and sometimes these results are not only imprecise but also incorrect.

A good example of the limitation of floating-point numbers and arithmetic was introduced by Rump [181], and it is often cited and studied (for example, in [45, 87]). It consists in computing the value of a function $f(a, b)$.

Example 3.0.1 *Consider the following function:*

$$f(a,b) = 333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b} \qquad (3.1)$$

with $a = 77617$ *and* $b = 33096$*. The results of computing* f *on an IBM S/370 main frame using single, double and extended-precision arithmetic are:*

Single precision:	$f = 1.172603...$
Double precision:	$f = 1.1726039400531...$
Extended precision:	$f = 1.172603940053178...$

It may seem that a reliable approximation of f is 1.172603 (using the argument that increasing the number of digits the result does not change), but the correct result of f is near to -0.827396, and it can be obtained using an arithmetic that considers 37 or more digits. Notice that even in the last case there is no guarantee in the result.

A modified version of this example for computers that use IEEE-754 arithmetic [4] can be found in [134]. This new expression produces the same effect as that proposed by Rump in a modern machine like Intel x86:

$$f(a,b) = (333.75 - a^2)b^6 + a^2(11a^2b^2 - 121b^4 - 2) + 5.5b^8 + \frac{a}{2b} \tag{3.2}$$

There are many good examples of the consequences of the different errors introduced when floating-point arithmetics is considered. One of them is the Patriot missile battery that failed to intercept a Scud missile in Saudi Arabia[1] (see [92, 176], for more examples).

The conclusion of all these examples is that floating-point arithmetic is not reliable, and it is necessary to use another method in order to obtain certified results.

Interval Analysis is a tool for automatically confining errors in numerical computation on machines. The seminal book in interval analysis was written by Moore [153] in 1966, and it is the main reference in the subject. Nevertheless, some of these ideas can be found in the work of Young [212] (thirty years before), who introduced the concept of *many-valued quantities*, and defined operations between these *quantities*. In her work, Young identifies the *lower value* ($\underline{a}$) and *upper value* ($\overline{a}$) of a quantity $\mathbf{a}$, and computes the bounds of the four basic operations $\{+, -, \times, \div\}$ in the same way that will be computed for intervals.

At the end of the fifties, three mathematicians developed in almost simultaneous way, the essential ideas of interval arithmetics: Warmus [210] in Poland, Sunaga [195] in Japan, and Moore [151, 152] in the United States; but the Moore's version has been the most influential. The aim of this method is to compute reliable results even when uncertainties in the data input, truncation or rounding operations are involved. It is based on a *new kind of number* called *interval*, represented by a pair of real numbers.

There are many well-known reference books [2, 87, 153, 155, 159], papers and reports [25, 92, 107, 176] about classic interval analysis, interval arithmetic and applications. There exists also a website with tutorials, papers and material (see [124, 141]).

The aim of this chapter is to present the basis of interval analysis and arithmetic, and some of the most important results that will be used later in this document. It is organized as follows: Section 3.1 presents a brief introduction to

[1] `http://www.fas.org/spp/starwars/gao/im92026.htm`

intervals (from an ideal point of view), while section 3.2 introduces classic interval arithmetic and its properties. Section 3.3 explains some differences between the operations *union* and *intersection* defined for *set of numbers* and for *interval numbers*. Section 3.4 introduces the concepts of *interval extensions* and *interval functions*. In Section 3.5, some methods to solving systems of equations using intervals are presented. Section 3.6 presents some details about implementations of interval arithmetic, while Section 3.7 give a brief introduction to generalized and modal intervals, an special extension of classic interval analysis.

3.1 Intervals

This section presents classic interval arithmetic from a theoretical point of view. That means, it does not consider the limitation of a machine or the representation of the real numbers. Anyway, section 3.6 presents some differences between this ideal point of view and the practical results obtained when a computational implementation of interval arithmetic is used. More details on these differences can be found in [97].

An interval $\mathbf{a}$ is the closed, connected subset of real numbers defined by

$$\mathbf{a} = [\underline{a}, \overline{a}] = \{x \in \mathbb{R} \mid \underline{a} \leq x \leq \overline{a}\} \tag{3.3}$$

where $\underline{a}, \overline{a} \in \mathbb{R}$, and $\underline{a} \leq \overline{a}$. An interval whose endpoints are the same, is called a *degenerate* interval, and can be identified with the *real* number equal to its identical endpoints. So, the interval $[5, 5]$ can be represented by the real number 5. Two intervals $\mathbf{x}$ and $\mathbf{y}$ are equal if their corresponding endpoints are equal. Thus,

$$\mathbf{x} = \mathbf{y} \iff (\underline{x} = \underline{y}) \text{ and } (\overline{x} = \overline{y}) \tag{3.4}$$

The set of intervals with bounds in $\mathbb{R}$ is denoted by $\mathbb{IR}$, and it is partially ordered by set inclusion.

$$\mathbf{x} \subseteq \mathbf{y} \iff (\underline{x} \geq \underline{y}) \text{ and } (\overline{x} \leq \overline{y}) \tag{3.5}$$

Example 3.1.1 *Consider the intervals* $\mathbf{x} = 5$ *(degenerated interval),* $\mathbf{y} = [5, 5]$, $\mathbf{z} = [4, 5]$, *and* $\mathbf{w} = [0, 10]$. *They can be ordered by set inclusion as following:*

$$\mathbf{x} \subseteq \mathbf{y} \subseteq \mathbf{z} \subseteq \mathbf{w} \quad = \quad 5 \subseteq [5, 5] \subseteq [4, 5] \subseteq [0, 10]$$

Here are some concepts related with intervals that will be useful thereafter: the *midpoint* $m(\mathbf{x})$, the *radius* $r(\mathbf{x})$, the *magnitude* $|\mathbf{x}|$, and the *width* $w(\mathbf{x})$.

$$\begin{aligned} m(\mathbf{x}) &= \frac{1}{2}(\underline{x} + \overline{x}) \\ r(\mathbf{x}) &= \frac{1}{2}(\overline{x} - \underline{x}) \\ |\mathbf{x}| &= \max_{x \in \mathbf{x}} |x| = \max\{|\underline{x}|, |\overline{x}|\} \\ w(\mathbf{x}) &= \overline{x} - \underline{x} = 2r(\mathbf{x}) \end{aligned} \tag{3.6}$$

Example 3.1.2 *Consider the interval* $\mathbf{x} = [-3, 1]) \in \mathbb{IR}$. *Then*

$$m(\mathbf{x}) = -1, \quad r(\mathbf{x}) = 2, \quad |\mathbf{x}| = 3, \quad w(\mathbf{x}) = 4$$

Notice that if $\mathbf{x} = [\underline{x}, \overline{x}]$ *is a degenerated interval, then* $m(\mathbf{x}) = \underline{x} = \overline{x}$ *and* $w(\mathbf{x}) = r(\mathbf{x}) = 0$.

An interval vector $\mathbf{x} \in \mathbb{IR}^n$, is an ordered n-tuple of intervals $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. It can be represented by a n-dimensional rectangle of points $(a_i, b_i), i = 1, \ldots, n$ such that $a_i = \underline{x}_i$ and $b_i = \overline{x}_i$. For this reason, an interval vector is often called *a box*. For example, the rectangle in figure 3.1 represents the box $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{IR}^2$.

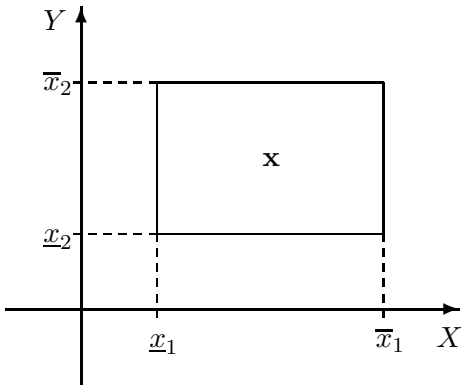


Figure 3.1: A graphic representation of an interval vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$.

The equality and inclusion defined in (3.4) and (3.5) are extended to interval vectors in the natural way, that is componentwise: $\mathbf{x} = \mathbf{y} \iff \forall i(\mathbf{x}_i = \mathbf{y}_i)$ and $\mathbf{x} \subseteq \mathbf{y} \iff \forall i(\mathbf{x}_i \subseteq \mathbf{y}_i)$. The first three concepts defined in (3.6) are extended in

the same way for an interval vector $\mathbf{x} \in \mathbb{IR}^n$:

$$\begin{aligned} m(\mathbf{x})_i &= m(\mathbf{x}_i) \\ r(\mathbf{x})_i &= r(\mathbf{x}_i) \\ |\mathbf{x}|_i &= |\mathbf{x}_i| \end{aligned} \tag{3.7}$$

They correspond to vectors in $\mathbb{R}^n$. The *width* $w(\mathbf{x}) \in \mathbb{R}^+$ and the *norm* $\|\mathbf{x}\| \in \mathbb{R}^+$ of an interval vector are scalars (real numbers) defined as following:

$$\begin{aligned} w(\mathbf{x}) &= 2\|r(\mathbf{x})\|_\infty = \max_i\{w(\mathbf{x}_i)\} \\ \|\mathbf{x}\| &= \||\mathbf{x}|\|_\infty = \max_i|\mathbf{x}_i| \end{aligned} \tag{3.8}$$

Example 3.1.3 *Consider the interval vector* $\mathbf{x} = ([-1,1],[5,9]) \in \mathbb{IR}^2$. *Then*

$$m(\mathbf{x}) = (0,7), \quad r(\mathbf{x}) = (1,2), \quad |\mathbf{x}| = (1,9), \quad w(\mathbf{x}) = 4, \quad \|\mathbf{x}\| = 9$$

An interval matrix $\mathbf{A} \in \mathbb{IR}^{m\times n}$, is an object with m rows and n columns of interval elements $\mathbf{a}_{i,j} \in \mathbb{IR}, i = 1,\ldots,m; j = 1,\ldots,n$. The equality and inclusion defined in (3.4) and (3.5) and the first three concepts in (3.6) are extended to interval matrices in the same way that they were extended to interval vectors (componentwise). The scalars defined in (3.8) are extended to matrices as follows:

$$\begin{aligned} w(\mathbf{A}) &= \max_{i,j}\{w(\mathbf{a}_{i,j})\} \\ \|\mathbf{A}\| &= \||\mathbf{A}|\|_\infty = \max_i\{\sum_j|\mathbf{a}_{i,j}|\} \end{aligned} \tag{3.9}$$

Example 3.1.4 *Consider the interval matrix* $\mathbf{A} = \begin{pmatrix} [0,2] & [-1,1] \\ [2,4] & [-1,3] \end{pmatrix} \in \mathbb{IR}^{2\times 2}$. *Then*

$$m(\mathbf{A}) = \begin{pmatrix} 1 & 0 \\ 3 & 2 \end{pmatrix}, r(\mathbf{A}) = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}, |\mathbf{A}| = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}, w(\mathbf{A}) = 4, \|\mathbf{A}\| = 7$$

Let $\mathbf{y} \in \mathbb{IR}^m, \mathbf{A} \in \mathbb{IR}^{m\times n}, \mathbf{x} \in \mathbb{IR}^n$, and $\mathbf{y} = \mathbf{A}\mathbf{x}$. It can be proved that $\|\mathbf{y}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{x}\|$ (a property equivalent to infinity norm of real vectors and matrices).

3.2 Interval Arithmetic

As noted by Moore [155], intervals have a *dual nature*, that means they can be considered as a *number* represented by the ordered pair of their endpoints (like a complex number $a + ib$ is represented by a and b) or a *set of real numbers*.

Considering intervals as *sets of numbers*, the result of an operation between two sets $\mathbf{x}$ and $\mathbf{y}$ must be a new set containing all possible combinations among an element from $\mathbf{x}$ and an element from $\mathbf{y}$. This is the most important property in interval arithmetic and is called *correctness*. Therefore, if $\diamond$ represents any of the four basic operators $\{+, -, \times, /\}$, the corresponding interval operation is defined as:

$$\mathbf{x} \diamond \mathbf{y} = \{x \diamond y \mid \ x \in \mathbf{x} \ \wedge \ y \in \mathbf{y}\} \tag{3.10}$$

with $\mathbf{x}/\mathbf{y}$ undefined if $0 \in \mathbf{y}$. Notice that the constraint $0 \notin \mathbf{y}$ assures that the four basic operations (if defined) are closed in $\mathbb{IR}$. Anyway, several authors have pointed out the importance of a definition for the division by an interval containing zero (mainly for using with interval Newton like methods). The first contribution was done by Kahan [109] in 1968, but only in the nineties it is possible to find formal definitions with proofs of correctness (with the works of Hansen [87], Ratz [171], Walster [207], and Hickey [97]).

Notice that from definition (3.10) it follows that

$$\mathbf{x} \subseteq \mathbf{y} \ \text{ and } \ \mathbf{z} \subseteq \mathbf{w} \ \Rightarrow \ \mathbf{x} \diamond \mathbf{z} \ \subseteq \ \mathbf{y} \diamond \mathbf{w} \tag{3.11}$$

That is another important property in interval arithmetic, and it is called *inclusion monotony*. Thus, the four basic operations are inclusion monotonic.

Although definition (3.10) is not operational, it is possible to express the results based on the bounds of the involved interval as following:

$$\begin{aligned}
\mathbf{x} + \mathbf{y} &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \\
\mathbf{x} - \mathbf{y} &= [\underline{x} + \overline{y}, \overline{x} - \underline{y}], \\
\mathbf{x} \times \mathbf{y} &= [min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}] \\
1/\mathbf{y} &= [1/\overline{y}, 1/\underline{y}], \qquad 0 \notin \mathbf{y} \\
\mathbf{x}/\mathbf{y} &= \mathbf{x} \times 1/\mathbf{y} \qquad 0 \notin \mathbf{y}
\end{aligned} \tag{3.12}$$

Example 3.2.1 *Consider the intervals* $\mathbf{x} = [1, 2]$, $\mathbf{y} = [-3, 1]$, *and* $\mathbf{z} = [-5, -4]$.

Then

$$\mathbf{x}+\mathbf{y} = [1,2]+[-3,1] = [-2,3] \qquad \mathbf{x}-\mathbf{y} = [1,2]-[-3,1] = [0,5]$$
$$\mathbf{y}\times\mathbf{z} = [-3,1]\times[-5,-4] = [-5,15] \qquad \mathbf{y}\div\mathbf{z} = [-3,1]\div[-5,-4] = [-\tfrac{1}{4},\tfrac{3}{4}]$$

It is important to notice that the interval substraction is not the inverse of interval addition, and the interval division is not the inverse of interval multiplication (in the sense of its real counterpart). That means the expressions $\mathbf{x}+\mathbf{y}=\mathbf{z}$, $\mathbf{x}=\mathbf{z}-\mathbf{y}$, and $\mathbf{y}=\mathbf{z}-\mathbf{x}$ are not equivalent.

Example 3.2.2 *Consider the intervals* $\mathbf{x}=[3,5]$, $y=[2,3]$, *and* $\mathbf{z}=[5,8]$. *Then*

$$\begin{aligned} \mathbf{x}+\mathbf{y} &= [3,5]+[2,3] &&= [5,8] \\ \mathbf{z}-\mathbf{x} &= [5,8]-[3,5] &&= [0,5] \\ \mathbf{z}-\mathbf{y} &= [5,8]-[2,3] &&= [2,6] \end{aligned}$$

Moreover, the expression $\mathbf{x}+\mathbf{y}-\mathbf{y}$ is equal to $\mathbf{x}$ only when $\mathbf{y}$ is a degenerated interval. The value of $\mathbf{y}-\mathbf{y}$ is in general different to zero. It may be difficult to understand that a number $\mathbf{y}$ minus itself is not zero, but if you think about intervals as set of numbers, the question *What is the result of take an element* y_1 *from* $\mathbf{y}$ *and substract another element* y_2 *from* $\mathbf{y}$*?*, has more than one answer. This situation, called *decorrelation* (or dependency problem), is considered as a drawback of interval arithmetic. When an interval appears more than once in an expression, each occurrence becomes an independent set of values. Subdistributy property and the absence of inverse element in section 3.2.1 are consequences of this drawback.

Computations with intervals are not reduced to the four basic operations. It can be generalized, for example, for unary operations[2] (like *sqr*, *sin*, *cos*, etc). If $r(x)$ is a continuous unary operation on $\mathbb{R}$, then

$$r(\mathbf{x}) = [\min_{x\in\mathbf{x}} r(x), \max_{x\in\mathbf{x}} r(x)] \tag{3.13}$$

defines a (subordinate) unary operation on $\mathbb{IR}$. Operational definition may also be given in the case of unary operations or classic binary operations. For example,

[2]Many authors prefer to use the name *elementary function* instead of *unary operation*. Both terms are considered *equivalent* for this presentation.

consider the operation *power* applied to an interval $\mathbf{x}$. Then

$$\mathbf{x}^n = \begin{cases} [1,1] & \text{if } n = 0 \\ [\underline{x}^n, \overline{x}^n] & \text{if } \underline{x} \geq 0 \text{ and } n \text{ is even, or if } n \text{ is odd} \\ [\overline{x}^n, \underline{x}^n] & \text{if } \overline{x} \leq 0 \text{ and } n \text{ is even} \\ [0, max(\underline{x}^n, \overline{x}^n)] & \text{if } \underline{x} \leq 0 \leq \overline{x} \text{ and } n \text{ is even} \end{cases}$$

Example 3.2.3 *Consider the intervals* $\mathbf{x} = [-1, 5]$ *and* $\mathbf{y} = [\frac{\pi}{4}, \pi]$. *Then*

$$\begin{array}{ll} \mathbf{x}^2 = [-1,5]^2 = [0,25] & sin(\mathbf{y}) = sin([\frac{\pi}{4}, \pi]) = [0,1] \\ \mathbf{x}^3 = [-1,5]^3 = [-1,125] & cos(\mathbf{y}) = cos([\frac{\pi}{4}, \pi]) = [-1, \frac{\sqrt{2}}{2}] \end{array}$$

3.2.1 Properties of Interval Arithmetic

The following algebraic properties are consequences of the definitions of the interval arithmetic operations (3.10).

1. Commutativity:
   $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$
   $\mathbf{x} \times \mathbf{y} = \mathbf{y} \times \mathbf{x}$

2. Associativity:
   $(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$
   $(\mathbf{x} \times \mathbf{y}) \times \mathbf{z} = \mathbf{x} \times (\mathbf{y} \times \mathbf{z})$

3. Neutral element:
   $\mathbf{x} = \mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x} \quad \forall \mathbf{x} \in \mathbb{IR} \iff \mathbf{y} = [0,0]$
   $\mathbf{x} = \mathbf{x} \times \mathbf{y} = \mathbf{y} \times \mathbf{x} \quad \forall \mathbf{x} \in \mathbb{IR} \iff \mathbf{y} = [1,1]$

4. Elimination laws:
   $\mathbf{x} + \mathbf{y} = \mathbf{x} + \mathbf{z} \iff \mathbf{y} = \mathbf{z}$
   $\mathbf{x} \times \mathbf{y} = \mathbf{x} \times \mathbf{z} \quad \text{with } \underline{x}, \overline{x} > 0 \iff \mathbf{y} = \mathbf{z}$

5. $\mathbb{IR}$ has no zero divisor:
   $\mathbf{x} \times \mathbf{y} = 0 \iff (\mathbf{x} = 0 \text{ or } \mathbf{y} = 0)$

6. There is no inverse element for $+$ and $\times$ when $\mathbf{x} \in \mathbb{IR}$ with $\underline{x} \neq \overline{x}$. Nevertheless it follows that:
   $0 \in \mathbf{x} - \mathbf{x} \quad \text{and} \quad 1 \in \mathbf{x}/\mathbf{x} \quad (0 \notin \mathbf{x})$
   $\mathbf{x} + \mathbf{y} = 0 \implies \mathbf{x} \text{ and } \mathbf{y} \text{ are degenerated.}$
   $\mathbf{x}/\mathbf{y} = 1 \implies \mathbf{x} \text{ and } \mathbf{y} \text{ are degenerated and not equal to zero.}$

7. Subdistributivity:
   $\mathbf{x} \times (\mathbf{y} + \mathbf{z}) \subseteq \mathbf{x} \times \mathbf{y} + \mathbf{x} \times \mathbf{z}$
   $x \times (\mathbf{y} + \mathbf{z}) = x \times \mathbf{y} + x \times \mathbf{z} \qquad x \in \mathbb{R}$ and $\mathbf{y}, \mathbf{z} \in \mathbb{IR}$
   $\mathbf{x} \times (\mathbf{y} + \mathbf{z}) = \mathbf{x} \times \mathbf{y} + \mathbf{x} \times \mathbf{z} \qquad$ if $yz \geq 0$ for all $y \in \mathbf{y}, z \in \mathbf{z}$

3.3 Union and Intersection

This section is included because there are some important differences between the natural idea of *union* and *intersection*, and the operations *meet* and *join* used in the case of interval numbers. As shown in section 3.1, an interval can be considered as a *set of real numbers*. Using this interpretation, the intersection of two intervals $\mathbf{x}$ and $\mathbf{y}$ must be the set of real numbers included in both $\mathbf{x}$ and $\mathbf{y}$. In the other hand, the union of two intervals is the set of real numbers included either in $\mathbf{x}$ or $\mathbf{y}$.

Example 3.3.1 *Consider the intervals* $\mathbf{x} = [1, 4]$, $\mathbf{y} = [0, 2]$, *and* $\mathbf{z} = [3, 5]$. *Then*

$$\mathbf{x} \cap \mathbf{y} = [1, 2] \quad \textit{and} \quad \mathbf{x} \cup \mathbf{y} = [0, 4] \tag{3.14}$$
$$\mathbf{x} \cap \mathbf{z} = [3, 4] \quad \textit{and} \quad \mathbf{x} \cup \mathbf{z} = [1, 5] \tag{3.15}$$
$$\mathbf{y} \cap \mathbf{z} = \emptyset \quad \textit{and} \quad \mathbf{y} \cup \mathbf{z} = \{[0, 2], [3, 5]\} \tag{3.16}$$

These operations correspond to the *set intersection* and *set union*, but they are not closed in $\mathbb{IR}$ (notice that the results in (3.16) are not interval numbers). Instead of them, the *interval meet* and *join* operations define the intersection and union between intervals, respectively. Thus,

$$\begin{aligned} \mathbf{x} \wedge \mathbf{y} &= [max(\underline{x}, \underline{y}), min(\overline{x}, \overline{y})] \qquad \text{if } max(\underline{x}, \underline{y}) \leq min(\overline{x}, \overline{y}) \\ \mathbf{x} \vee \mathbf{y} &= [min(\underline{x}, \underline{y}), max(\overline{x}, \overline{y})] \end{aligned}$$

In other words, the interval *meet* between $\mathbf{x}$ and $\mathbf{y}$ represents the bigger interval $\mathbf{z} \in \mathbb{IR}$ such that $\mathbf{z} \subseteq \mathbf{x}$ and $\mathbf{z} \subseteq \mathbf{y}$, while the interval *join* represents the smaller interval $\mathbf{z} \in \mathbb{IR}$ such that $\mathbf{x} \subseteq \mathbf{z}$ and $\mathbf{y} \subseteq \mathbf{z}$. Both operations (if defined) are closed in $\mathbb{IR}$.

Moore's Diagram[3] of figure 3.2 shows the set of interval $\mathbf{z} \in \mathbb{IR}$ in which an interval $\mathbf{a}$ is included, while the diagram of 3.3 show the set of interval included in

[3] A Moore's Diagram is an easy and friendly way to graphically represent intervals. Each interval $\mathbf{a} = [\underline{a}, \overline{a}]$ is identified by a point $(\underline{a}, \overline{a})$ in the cartesian plane. The line $x = y$ represents the set of degenerated intervals, while the points (x, y) with $x \leq y$ represent $\mathbb{IR}$ (See [85, 84]).

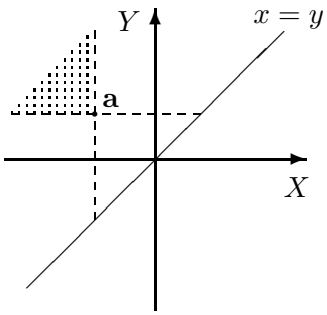


Figure 3.2: $\{\mathbf{z} \in \mathbb{IR} \mid \mathbf{z} \supseteq \mathbf{a}\}$.

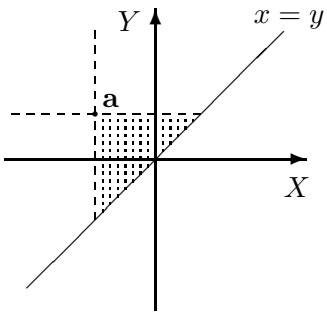


Figure 3.3: $\{\mathbf{z} \in \mathbb{IR} \mid \mathbf{z} \subseteq \mathbf{a}\}$.

a. Using this diagram it is easily to identify the operations *meet* and *join* between two or more intervals.

Figure 3.4 graphically shows the results of operations *meet* and *join* between the intervals **y** and **x** of the last example, while figure 3.5 shows the results for intervals **y** and **z**.

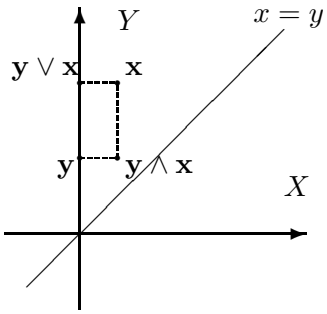


Figure 3.4: *Meet* and *join* of **y** and **x**.

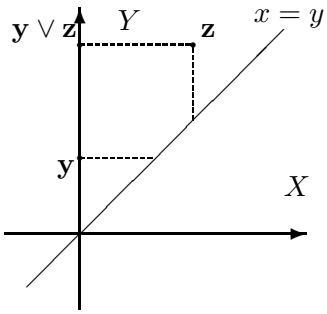


Figure 3.5: *Join* of **y** and **z**.

Notice that if $max(\underline{x}, \underline{y}) \leq min(\overline{x}, \overline{y})$ for two intervals **x** and **y**, the operations *meet* and *join* obtain the same result that the set *intersection* and set *union*, respectively.

3.4 Interval Functions

Let f be a continuous real function over a box **x**. The image of **x** under the mapping f (that means, the set of values $f(x)$ with $x \in \mathbf{x}$) is denoted by

$$range(f, \mathbf{x}) = \{f(x) \mid x \in \mathbf{x}\} \tag{3.17}$$

A general property of continuous functions is that they map connected sets

into connected sets (and compact sets into compact sets), so the real arithmetic operations map intervals into closed intervals. Notice that the ranges of the four elementary interval arithmetic operations are exactly the ranges of the corresponding real operations. Although in some particular cases the range of a function f may be easily computable, in a general case it is not. Hence the goal of interval arithmetic is to find a computable interval extension $\mathbf{f}$ of a function f, such that for all $x \in \mathbf{x}$, $f(x) \in \mathbf{f}(\mathbf{x})$.

Definition 6 (Interval Extension) *Let f be a continuous real function f of n real variables $x_1, \ldots, x_n$. The interval function $\mathbf{f}$ of n intervals $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is an interval extension of f iff*

$$\mathbf{f}(x_1, \ldots, x_n) = f(x_1, \ldots, x_n) \qquad \forall x \in \mathbb{R} \tag{3.18}$$

That means, if the arguments are degenerated intervals the result of computing $\mathbf{f}(x_1, \ldots, x_n)$ is the interval $[f(x_1, \ldots, x_n), f(x_1, \ldots, x_n)]$.

Example 3.4.1 *Consider the real function $f(x) = x^2 - 2x$. The following interval functions are interval extensions of f.*

$$\begin{array}{ll} \mathbf{f}_1(\mathbf{x}) = \mathbf{x}^2 - 2\mathbf{x} & \mathbf{f}_2(\mathbf{x}) = \mathbf{x}(\mathbf{x} - 2) \\ \mathbf{f}_3(\mathbf{x}) = \mathbf{x}^2 - 3\mathbf{x} + \mathbf{x} & \mathbf{f}_4(\mathbf{x}) = \mathbf{x}\mathbf{x} - \mathbf{x} - \mathbf{x} \end{array}$$

As is shown in the example, there is never a unique interval extension of a given real function f, but an interval extension $\mathbf{f}$ will be better (in the sense of interval inclusion) than an interval extension $\mathbf{g}$ iff $\mathbf{f}(\mathbf{x}) \subseteq \mathbf{g}(\mathbf{x})$. In the last example, if $\mathbf{x} = [-1, 2]$ then $\mathbf{f}_1(\mathbf{x}) = [-4, 6]$, $\mathbf{f}_2(\mathbf{x}) = [-6, 3]$, $\mathbf{f}_3(\mathbf{x}) = [-7, 9]$, and $\mathbf{f}_4(\mathbf{x}) = [-6, 6]$. Of course, the $range(f, \mathbf{x}) = [-1, 3]$ is included in all results of the interval extensions.

Definition 7 (United Extension) *Given a continuous real function f of n real variables with interval domains $\mathbf{x}_1, \ldots, \mathbf{x}_n$, the* united extension *R_f is defined as the range of function values*

$$R_f(\mathbf{x}) = [\min_{x \in \mathbf{x}} f(x), \max_{x \in \mathbf{x}} f(x)] \tag{3.19}$$

where $x = (x_1, \ldots, x_n)$ and $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

Although the united extension of a function f is its optimal interval extension, it is not computable in general. For this reason, one of the most important

objective of interval analysis is to provide a computable interval extension of a function.

It is important to note that an interval function is defined by the rational[4] expression given for it, and this expression cannot be changed without changing the associated interval function. That means, the expression $\mathbf{x}+1/\mathbf{x}$ (in the variable $\mathbf{x}$) is not *equivalent* to the expression $(\mathbf{x}^2+1)/\mathbf{x}$ (even though $x+1/x = (x^2+1)/x$ in real arithmetic). This is a direct consequence of the dependency problem presented in section 3.2.

Theorem 1 *if* $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ *is a rational expression in the interval variables* $\mathbf{x}_1, \ldots, \mathbf{x}_n$*, then*

$$\mathbf{x}'_1 \subseteq \mathbf{x}_1, \ldots, \mathbf{x}'_n \subseteq \mathbf{x}_n \quad \Longrightarrow \quad \mathbf{f}(\mathbf{x}'_1, \ldots, \mathbf{x}'_n) \subseteq \mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n) \tag{3.20}$$

for every set of interval numbers $\mathbf{x}_1, \ldots, \mathbf{x}_n$ *for which the interval arithmetic operations in* $\mathbf{f}$ *are defined.*

This theorem introduced by Moore (see theorem 3.1 in [153]) is considered as *the fundamental theorem of interval arithmetic*, and it allows interval arithmetic to obtain certified results when implemented on a machine. This theorem is related with a property called *inclusion monotonic* and defined as follows:

Definition 8 (Inclusion Monotonic) *Let* $\mathbf{f}$ *be an interval valued function of the interval variables* $\mathbf{x}_1, \ldots, \mathbf{x}_n$*.* $\mathbf{f}$ *is inclusion monotonic if*

$$\mathbf{x}'_1 \subseteq \mathbf{x}_1, \ldots, \mathbf{x}'_n \subseteq \mathbf{x}_n \quad \Longrightarrow \quad \mathbf{f}(\mathbf{x}'_1, \ldots, \mathbf{x}'_n) \subseteq \mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n) \tag{3.21}$$

As is shown in section 3.2, the four basic operators are inclusion monotonic. Moreover, all unary operations presented in (3.13) are inclusion monotonic too.

The importance of the theorem introduced by Moore is that every function based on a rational expression in a set of interval variables is inclusion monotonic. In particular, the interval extension of a function f, obtained by substituting each real variable by the corresponding interval variable and the real operations by the corresponding interval operations, is inclusion monotonic. This extension is called *the natural interval extension of the function* f and it was introduced by Moore [155].

[4] A *rational* expression in the variables $x_1, \ldots, x_n$ means a finite combination of $x_1, \ldots, x_n$ and a finite set of constants with arithmetic operations such that the expression has a sense.

Example 3.4.2 *Consider the real functions* $f_1(x) = x^2 - 2x$ *and* $f_2(x) = \sqrt{x-2}$, *and their natural interval extensions* $\mathbf{f}_1(\mathbf{x}) = \mathbf{x}^2 - 2\mathbf{x}$ *and* $\mathbf{f}_2(\mathbf{x}) = \sqrt{\mathbf{x}-2}$. *Related to the intervals* $\mathbf{x}_1 = [3,6] \subseteq \mathbf{x}_2 = [2,11]$, *it follows that*

$$R_{f_1}(\mathbf{x}_1) = [3,24] \subseteq \mathbf{f}_1(\mathbf{x}_1) = [-3,30] \tag{3.22}$$

$$\mathbf{f}_1(\mathbf{x}_1) = [-3,30] \subseteq \mathbf{f}_1(\mathbf{x}_2) = [-18,117] \tag{3.23}$$

$$R_{f_2}(\mathbf{x}_1) = [1,2] \subseteq \mathbf{f}_2(\mathbf{x}_1) = [1,2] \tag{3.24}$$

$$\mathbf{f}_2(\mathbf{x}_1) = [1,2] \subseteq \mathbf{f}_2(\mathbf{x}_2) = [0,3] \tag{3.25}$$

Notice that in general, the natural interval extension of a function f is an overestimation of the range of f. Nevertheless, if $f(x_1, \ldots, x_n)$ is based on a rational expression in which each variable x_i occurs only once[5], then its natural interval extension $\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ computes the actual range of values for $x_i \in \mathbf{x}_i$. That means

$$\mathbf{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \{f(x_1, \ldots, x_n) \mid x_i \in \mathbf{x}_i, \quad \forall i = 1, \ldots, n\} \tag{3.26}$$

As one of the central problem in Interval Analysis is to provide sharp estimations of the range of a function and the natural interval extension is optimal only for particular cases (single occurrence of variables), other interval extensions have been proposed.

For example, *mean value interval extension* is based on the Mean Value Theorem which states that if a function $f(x)$ is defined and continuous on the interval $[a,b]$ and differentiable on $]a,b[$, then there is at least one value $c \in]a,b[$ such that

$$f'(c) = \frac{f(b) - f(a)}{b - a} \tag{3.27}$$

Notice that if equation (3.27) is true, then the following is also true:

$$f(x) = f(\tilde{x}) + f'(c)(x - \tilde{x})$$

where c is a point between x and $\tilde{x}$ with $x, \tilde{x} \in [a,b]$. Therefore, for a given interval $\mathbf{x}$, the last expression can be used for approximating the range of a continuous differentiable function $f(x)$ in the following way:

$$R_f(\mathbf{x}) \subseteq f(\tilde{x}) + \mathbf{f}'(\mathbf{x})(\mathbf{x} - \tilde{x})$$

[5]Notice that the expression $x \cdot x$ is considered as a double occurrence of the variable x.

where $\tilde{x} \in \mathbf{x}$ (generally $\tilde{x} = m(\mathbf{x})$ the middle point of the interval) and $\mathbf{f}'(\mathbf{x})$ is an interval containing all possible values $f'(x)$ in the interval $\mathbf{x}$. Obviously, the problem now is to compute a sharp approximation of $\mathbf{f}'(\mathbf{x})$. An easy way (but not so sharp) to compute an enclosure of $\mathbf{f}'(\mathbf{x})$ is to use the natural interval extension of $f'(x)$. Another method to approximate $\mathbf{f}'(\mathbf{x})$ based on the slope function [172] can be found in [122, 156].

Another method to compute the range $R_f(\mathbf{x})$ for a given interval $\mathbf{x}$ can be obtained by using interval Taylor form [153, 179]. The main idea is to express the function $f(x)$ in the form:

$$f(x) = f(\tilde{x}) + \sum_{k=1}^{n} \frac{f^{(k)}(\tilde{x})}{k!}(x - \tilde{x})^k + \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - \tilde{x})^{(n+1)} \tag{3.28}$$

and to compute an enclosure of the error term $\frac{f^{(n+1)}(\xi)}{(n+1)!}(x - \tilde{x})^{(n+1)}$. For a given interval $\mathbf{x}$, the range of $f(x)$ is then approximated by:

$$R_f(\mathbf{x}) = f(\tilde{x}) + \sum_{k=1}^{n} \frac{f^{(k)}(\tilde{x})}{k!}(\mathbf{x} - \tilde{x})^k + \frac{\mathbf{f}^{(n+1)}(\mathbf{x})}{(n+1)!}(\mathbf{x} - \tilde{x})^{(n+1)} \tag{3.29}$$

where $\tilde{x} \in \mathbf{x}$ and $\mathbf{f}^{(n+1)}(\mathbf{x})$ is an interval containing all the values of $f^{(n+1)}(x)$ for the given interval $\mathbf{x}$. Obviously, it must be possible to compute the n derivatives of f in the point $\tilde{x}$ and a sharp enclosure of the error term. More details about Taylor forms interval extensions can be found in [160]. A good comparison between Taylor Forms and other types interval extensions can be found in [193].

3.5 Solving Equations Systems

Interval Analysis can be used not only for reliability reason in computational operations. The most important advances in Interval Analysis have been done in problem solving and global optimization. This section presents in a brief way some techniques for problem solving based on Interval Analysis. Most of the techniques described in this section can be found in [90, 107, 159].

This thesis focuses only on the resolution of systems of non-linear equations (in particular, distance equations systems). It is important to note that many of the techniques presented hereafter are also related to global optimization because the classic method to find the maximum and minimum of a function f is based on a system of equations of the form $f' = 0$. Other works related to global optimization

can be found in [161].

3.5.1 An Evaluation/Bisection Algorithm

One of the simplest approach to solving a system of non-linear equations using intervals is based on a combination of *evaluation* and *bisection* phases. The base algorithm *Evaluation/Bisection* is similar to the *Branch and Bound* algorithm commonly used in optimization.

Consider for example an equation $f(x) = 0$, an interval domain $\mathbf{x}$, and $\mathbf{f}(\mathbf{x})$ (an interval extension of f evaluated in $\mathbf{x}$). Following the properties of interval analysis (presented in Section 3.4), we know that $0 \notin \mathbf{f}(\mathbf{x})$ implies the equation has no solution in $\mathbf{x}$. Therefore, $0 \in \mathbf{f}(\mathbf{x})$ is a necessary condition for solving the system $f(x) = 0$. Algorithm 4 presents a procedure for solving a system using this condition.

Algorithm 4 Evaluation_Bisection($\mathbf{f}, \mathbf{x}, \epsilon$)

```
Q ← x
Sols ← ∅
while Q not empty do
  select and delete an interval x' from Q
  if 0 ∈ f(x') then
    if width(x') < ε then
      Sols ← Sols ∪ x'
    else
      Q ← Q ∪ bisection(x')
return Sols
```

This procedure evaluates the interval function $\mathbf{f}$ in a domain $\mathbf{x}'$. If the condition $0 \in \mathbf{f}(\mathbf{x}')$ is verified and the width of $\mathbf{x}'$ is less than the precision ϵ, the interval is stored in $Sols$ as a potential solution else the interval $\mathbf{x}'$ is bisected into two new intervals and both are added to the queue Q for posterior revision. The algorithm returns $Sols$ (the set of potential solutions).

This procedure is easily generalized for a system of m equations $f(x) = 0$ with $f = (f_1, \ldots, f_m)$ and $x = (x_1, \ldots, x_n)$ by considering the condition $(0, \ldots, 0) \in (\mathbf{f}_1(\mathbf{x}), \ldots, \mathbf{f}_m(\mathbf{x}))$ and bisecting the box $\mathbf{x}$ into two new boxes. The final result is a set of boxes with a given precision.

Example 3.5.1 *Consider the following system of two non-linear equations with*

initial domains $\mathbf{x}_1 = [0, 10]$, $\mathbf{x}_2 = [0, 10]$*):*

$$\begin{aligned} x_1 \cdot sin(x_2) - cos(x_1) \cdot x_2 - 5 &= 0 \\ x_1^2 + cos(x_2) - 8 &= 0 \end{aligned}$$

Table 3.1 shows the first iterations of Algorithm 4 applied to this system[6] *by considering* $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ *and* $\mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2)$*, with* $\mathbf{f}_1 = \mathbf{x}_1 \cdot sin(\mathbf{x}_2) - cos(\mathbf{x}_1) \cdot \mathbf{x}_2 - 5$ *and* $\mathbf{f}_2 = \mathbf{x}_1^2 + cos(\mathbf{x}_2) - 8$*. At the end of the process a set of boxes is returned.*

$\mathbf{x}'$	$0 \in \mathbf{f}(\mathbf{x}')$	*bisection*	Q
			$([0, 10], [0, 10]$
$([0, 10], [0, 10])$	yes	yes $(\mathbf{x}_1)$	$([0, 5], [0, 10]), ([5, 10], [0, 10])$
$([5, 10], [0, 10])$	no	–	$([0, 5], [0, 10])$
$([0, 5], [0, 10])$	yes	yes $(\mathbf{x}_2)$	$([0, 5], [0, 5]), ([0, 5], [5, 10])$
$([0, 5], [5, 10])$	yes	yes $(\mathbf{x}_1)$	$([0, 2.5], [5, 10]), ([2.5, 5], [5, 10]), ([0, 5], [0, 5])$
...	...	...	...

Table 3.1: First steps of Algorithm 4 applied to example 3.5.1.

It is important to note that the set of returned boxes includes all the solutions of the problem included in the initial domain (this is a safe procedure), but also that some boxes may contain no solution. For this reason, other techniques of numerical analysis are applied (as those presented in Section 3.5.3) in order to identify which boxes contain solutions.

One of the first works in this area, based on the *intermediate value theorem*[7] and the interval extension of a univariate real function f was introduced by Moore in [152] and corresponds to the following existence test (also known as *Rolle theorem*):

Theorem 2 *Let* $\mathbf{f}(\mathbf{x})$ *be an interval extension of* $f(x)$ *(with* $\mathbf{x} = [\underline{x}, \overline{x}]$*), and let* $\mathbf{f}'(\mathbf{x})$ *be the interval extension of* $f'(x) = \frac{df}{dx}$*. If*

$$f(\underline{x}) = \mathbf{f}(\underline{x}) < 0 < \mathbf{f}(\overline{x}) = f(\overline{x}) \quad \textit{and} \quad 0 \notin \mathbf{f}'(\mathbf{x}) \tag{3.30}$$

then there exists a unique solution x_{sol} *for* $f(x) = 0$ *in* $\mathbf{x}$*.*

[6] A graphic explanation of the procedure *evaluation+bisection* applied to this example can be found in the web site of the COPRIN project (see `http://www-sop.inria.fr/coprin`).

[7] This theorem states that if f is continuous on a closed interval $[a, b]$, and c is any number between $f(a)$ and $f(b)$ inclusive, then there is at least one number x in the closed interval such that $f(x) = c$.

Moreover, by the mean value theorem, if $\tilde{x} \in \mathbf{x}$ then

$$x_{sol} \in \tilde{x} - \frac{f(\tilde{x})}{\mathbf{f}'(\mathbf{x})} \tag{3.31}$$

and if the interval $\mathbf{x}$ is not too wide, the expression $\tilde{x} - \frac{\mathbf{f}(\tilde{x})}{\mathbf{f}'(\mathbf{x})}$ will be properly contained in $\mathbf{x}$ and the solution x_{sol} can be obtained with the following iterative process (which is the classical Newton-Raphson scheme)

$$\mathbf{x}_{n+1} = \tilde{x}_n - \frac{f(\tilde{x}_n)}{\mathbf{f}'(\mathbf{x}_n)} \tag{3.32}$$

with $\mathbf{x}_0 = \mathbf{x}$ and $\tilde{x}_n = m(\mathbf{x}_n)$.

Example 3.5.2 *Consider the real function $f(x) = x^2 - 2$ and the interval $\mathbf{x} = [1, 2]$. Let $\mathbf{f}(\mathbf{x}) = \mathbf{x}^2 - 2$ and $\mathbf{f}'(\mathbf{x}) = 2\mathbf{x}$ be the natural interval extension of $f(x)$ and $f'(x)$ respectively. Using (3.32) with $\mathbf{x}_0 = [1, 2]$, and $\tilde{x}_0 = 1.5$, it follows*

$$\begin{array}{ll}
\mathbf{x}_0 = [1, 2] & \tilde{x}_0 = 1.5 \\
\mathbf{x}_1 = [1.375, 1.4375] & \tilde{x}_1 = 1.40625 \\
\mathbf{x}_2 = [1.4140625, 1.414417613\ldots] & \tilde{x}_2 = 1.414240056\ldots \\
\mathbf{x}_3 = [1.414213559\ldots, 1.414213565\ldots] & \tilde{x}_3 = 1.414213562\ldots \\
\mathbf{x}_4 = [1.414213562\ldots, 1.414213562\ldots] & \tilde{x}_4 = 1.414213562\ldots
\end{array}$$

Notice that the solution of $f(x) = 0$ ($x_{sol} = \sqrt{2} = 1.414213562\ldots$) is contained in all intervals $\mathbf{x}_0, \ldots, \mathbf{x}_4$ (as it is attended).

Actually, the observation did by Moore is a clear application of the Brouwer's theorem. The expression given to compute the final solution is quite similar to the interval Newton method presented in Section 3.5.2.

3.5.2 Fixed Point based Methods

Informally, given a mapping $f : D_1 \to D_2$ an a value $x \in D_1$, x is called a fixed point of f if $f(x) = x$. In other words, the point x is not affected by the mapping.

Many methods for solving $f(x) = 0$ are based on iterative algorithms that look for a fixed point in a given domain. The existence of this fixed point is guarantee by the Brouwer's theorem stated as follows:

Theorem 3 *Let D be homeomorphic to the closed unit ball* [8] *in $\mathbb{R}^n$, and let*

[8] The closed unit ball is the set of all points in Euclidean n-space $\mathbb{R}^n$ which are at distance at most 1 from the origin.

$f : D \to D$ be a continuous mapping such that the range $f(D) \subseteq D$. Then f has a fixed point in D (i.e. there is a $x \in D$ such that $f(x) = x$).

In other words, the theorem states that every continuous function which maps elements from a closed domain into itself has at least one fixed point. This is the main argument of many existence and uniqueness theorems (see for example *Moore*[154] and *Miranda*[148]), and the base of others techniques for solving systems of non-linear equations (see [121] and [88, 89]).

Interval Newton Method

The main idea is to compute an enclosure of the zero of a real function $f(x)$ by iteratively improving an initial approximation $\mathbf{x} = [\underline{x}, \overline{x}]$. This method assumes that the derivative $f'(x)$ is continuous in $\mathbf{x}$ and that it is possible to compute an interval extension $\mathbf{f}'(\mathbf{x})$. Two additional conditions are needed: $0 \notin \mathbf{f}'(\mathbf{x})$ and $f(\underline{x}) \cdot f(\overline{x}) < 0$. Starting with $\mathbf{x}_0 = \mathbf{x}$, the method iterates as follows:

- $\tilde{x}_n = m(\mathbf{x}_n)$
- $N(\tilde{x}_n, \mathbf{x}_n) = \tilde{x}_n - \frac{f(\tilde{x}_n)}{\mathbf{f}'(\mathbf{x}_n)}$
- $\mathbf{x}_{n+1} = \mathbf{x}_n \cap N(\tilde{x}_n, \mathbf{x}_n)$

The value $\tilde{x}_n$ may be any value inside the interval $\mathbf{x}$ but some works suggest that the middle point of $\mathbf{x}$ is the best choice for it.

Example 3.5.3 *Consider the equation $f(x) = 0$ with*

$$f(x) = \sqrt{x} + (x + 1) \cdot cos(x) \tag{3.33}$$

and the initial domain $\mathbf{x} = [2, 3]$. Table 3.2 shows four iterations of the interval Newton method applied to equation (3.33) considering the natural interval extension

$$\mathbf{f}'(\mathbf{x}) = \frac{1}{2\sqrt{\mathbf{x}}} + cos(\mathbf{x}) - (\mathbf{x} + 1) \cdot sin(\mathbf{x})$$

Figure 3.6 graphically presents the first iteration of the method.

The result of $N(\tilde{x}_0, \mathbf{x}_0)$ is commonly called *interval Newton operator* and has some interesting properties that will be presented below.

In the case of systems of equations, the interval Newton operator is derived from a linearization of the system by considering an enclosure of its Jacobian

$\mathbf{x}_n$	$\tilde{x}_n$	$N(\tilde{x}_n, \mathbf{x}_n)$	$\mathbf{x}_{n+1}$
$[2, 3]$	2.5	$[-0.016421, 2.218137]$	$[2, 2.218137]$
$[2, 2.218137]$	2.109068	$[2.051401, 2.064726]$	$[2.051401, 2.064726]$
$[2.051401, 2.064726]$	2.058063	$[2.059037, 2.059053]$	$[2.059037, 2.059053]$
$[2.059037, 2.059053]$	2.059045	$[2.059045, 2.059045]$	$[2.059045, 2.059045]$

Table 3.2: First iterations of the interval Newton method applied to (3.33).

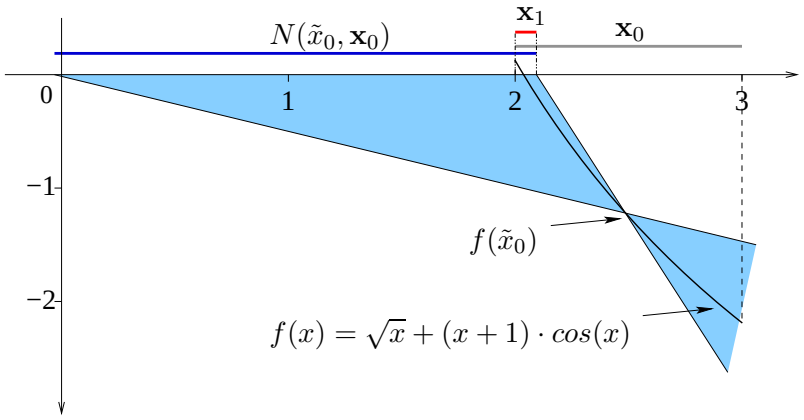


Figure 3.6: First iteration of interval Newton method ($\mathbf{x}_1 = N(\tilde{x}_0, \mathbf{x}_0) \cap \mathbf{x}_0$).

matrix. Consider a function $f : \mathbb{R}^n \to \mathbb{R}^n$ continuous and differentiable in a domain $D \subseteq \mathbb{R}^n$ (in particular $D = \mathbf{x} \in \mathbb{IR}^n$) and $\mathbf{J}^{-1}(\mathbf{x})$ as an interval matrix which contains all the matrix $J^{-1}(x)$ where J represents a Jacobian matrix of the system evaluated in x (with $x \in \mathbf{x}$). The new expression of the interval Newton operator (INO) for non-linear systems $f(x) = 0$ can be written as follows:

$$INO(\mathbf{x}) = \tilde{x} - \mathbf{J}^{-1}(\mathbf{x}) \cdot f(\tilde{x}) \tag{3.34}$$

All solutions of the system (if exist) will be contained in $INO(\mathbf{x})$. Obviously, this expression considers the regularity of the Jacobian matrix of the system (on the other case, $\mathbf{J}^{-1}(\mathbf{x})$ cannot be computed). The method iterates using the following expression:

$$\mathbf{x}_{n+1} = \mathbf{x}_n \cap (\tilde{x}_n - \mathbf{J}^{-1}(\mathbf{x}_n) \cdot f(\tilde{x}_n)) \tag{3.35}$$

where $\mathbf{x}_0 = \mathbf{x}$ (the original domain) and $\tilde{x}_i = m(\mathbf{x}_i)$.

The interval Newton operator has two important properties:

- If $INO(\mathbf{x}) \subseteq \mathbf{x}$ then there is a unique solution in $\mathbf{x}$.
- If $INO(\mathbf{x}) \cap \mathbf{x} = \emptyset$ then no solution exists in $\mathbf{x}$.

Important improvements to this basic operator have been proposed in [123] (the *Krawczyk operator*) and [89] (the *Hansen-Sengupta Algorithm*).

Krawczyk operator

The Krawczyk operator [123] is an alternative of the interval Newton method commonly used when the Jacobian matrix of the system is singular in at least one point inside the interval $\mathbf{x}$. The main idea is to use the center form to express $f(x)$ as follows:

$$f(x) = f(\tilde{x}) + J \cdot (x - \tilde{x}) \tag{3.36}$$

where J is the Jacobian matrix of the system evaluated in some point $\xi \in \mathbf{x}$. The solution of $f(x) = 0$ implies that $f(\tilde{x}) + J \cdot (x - \tilde{x}) = 0$. An algebraic manipulation is then performed to transform this expression into the following one:

$$x = \tilde{x} - f(\tilde{x}) + (I - J) \cdot (x - \tilde{x}) \tag{3.37}$$

where I represents the identity matrix. This is exactly the expression needed to find the roots of the system by computing a fixed point in the domain given by $\mathbf{x}$. It can be computed as follows:

$$\mathbf{x}_{n+1} = \tilde{x}_n - f(\tilde{x}_n) + (I - \mathbf{J}(\mathbf{x})) \cdot (\mathbf{x}_n - \tilde{x}_n) \tag{3.38}$$

with $\mathbf{x}_0 = \mathbf{x}$ (the initial domain) and $\mathbf{J}(\mathbf{x})$ is an interval matrix containing the jacobian matrix of the system evaluated in all the points $\xi \in \mathbf{x}$. A drawback of this method is that it narrows the domain $\mathbf{x}$ only if $(I - \mathbf{J}(\mathbf{x}))$ is near to zero (in other words, if $\mathbf{J}(\mathbf{x})$ is near to the identity matrix). For this reason the Krawczyk operator uses a matrix C (generally $C = m(\mathbf{J}(\mathbf{x}))^{-1}$) to precondition the system as follows:

$$K(\mathbf{x}) = \tilde{x}_n - Cf(\tilde{x}_n) + (I - C\mathbf{J}(\mathbf{x})) \cdot (\mathbf{x}_n - \tilde{x}_n) \tag{3.39}$$

As similar as INO the Krawczyk operator has some important properties as:

- If $K(\mathbf{x}) \cap \mathbf{x} = \emptyset$ then no solution exists in $\mathbf{x}$.
- If $K(\mathbf{x}) \subset \mathbf{x}$ then there is a unique solution in $\mathbf{x}$.

Notice that $K(\mathbf{x})$ needs to be completely included in $\mathbf{x}$ to prove the unicity of the solution.

3.5.3 Unicity Operator

One of the problems of the *Evaluation/Bisection* algorithm presented in Section 3.5.1 is that some result boxes may contain no solution. These boxes are returned because their widths are less than a given precision and no bisection is authorized. For this reason, other important tools are needed in order to prove the existence and/or uniqueness of a solution.

At the end of Section 3.5.1 a method to prove the existence of a solution for the equation $f(x) = 0$, proposed by Moore, has been presented. The operators of Section 3.5.2 can also be used to prove existence and uniqueness of solutions for systems of equations. This section presents another way to do it, based on the Kantorovich theorem[9] as presented in [143].

Theorem 4 *Let $f(x) = 0$ a system of n equations $f = (f_1, \ldots, f_n)$ and n unknowns $x = (x_1, \ldots, x_n)$. Let $\tilde{x}$ be a point for which the inverse of the Jacobian matrix of the system $J^{-1}(\tilde{x})$ exists, and let $\Omega = \{x \mid \|x - \tilde{x}\| \leq 2B\}$. If*

- $\|J^{-1}(\tilde{x})\| \leq A$,
- $\|J^{-1}(\tilde{x})f(\tilde{x})\| \leq 2B$,
- $\sum_{k=1}^{n} \left|\frac{\partial^2 f_i(x)}{\partial x_j \partial x_k}\right| \leq C$ *for* $i, j = 1, \ldots, n$ *and* $x \in \Omega$,
- A, B, C *constants satisfying* $2nABC \leq 1$.

then there is a unique solution $x^* \in \Omega$ such that $f(x^*) = 0$, and this solution can be obtained by using the Newton method with $\tilde{x}$ as initial estimation.

The advantage of the Kantorovich theorem is that it can be used not only for proving the uniqueness of a solution in a given domain $\mathbf{x}$, but also for inflating this initial domain in order to obtain a bigger zone containing only one solution (see [143, 182]). In [143], for example, Merlet shows a special case of inflation operation for distance equations which is able to compute the maximal box containing a sole solution (without using iterative methods). The key point of distance equations is that the expression $\frac{\partial^2 f_i(x)}{\partial x_j \partial x_k}$ becomes a constant which can be easily computed.

[9] The original presentation is discussed in [110, 196]. A comparison between these different tools can be obtained in [61]. Some explanations of how these tools are used when computational operations are involved can be obtained in [114, 181].

3.6 A Note about Implementation

When interval arithmetic is implemented in a machine, it uses the finite set of floating-point numbers that the hardware makes available [4, 70]. A simple real number is not always representable in a machine, so a floating-point number (usually the nearest) is instead used. Moreover, even if two numbers a and b are exactly representable in a machine, it is not always true that $a+b$ is representable too. This is another source of errors in computed results, and must be taken into account when interval arithmetic is implemented.

The standard IEEE 745 [4] (under revision since 2000, see [103]), defines formats for representing floating-point numbers (including ±zero, infinities, and NaNs), a set of floating-point operations, four rounding modes, and five exceptions. It was first implemented in the microprocessor Intel 8087 and it is a great support for implementations of interval arithmetic.

This section explains how interval arithmetic is implemented in a machine, and what types of operations are needed in order to guarantee the results. For this presentation, only the rounding modes will be of interest, specially the rounding *towards positive infinity* and *towards negative infinity* (also called rounding *upwards* and *downwards*).

Example 3.6.1 *Consider, for instance, a machine that can represent numbers with only three digits of mantissa. The value of* $\pi = 3.141596...$ *cannot be represented in this machine (actually, no machine can represent it), but keeping only three digits and rounding this number towards positive infinity the value* 3.15 *is a machine representable number. Using rounding towards negative infinity will give the value* 3.14*, and therefore the interval* $\mathbf{x} = [3.14, 3.15]$ *is a safe representation of the number* π *in this machine.*

In the last example, an interval $\mathbf{y} = [3, 4]$ is already a safe representation of the number π, but in general it is desirable to make interval bounds as narrow[10] as possible. In order to do it, the *hull approximation* of a set is defined.

Definition 9 (Hull approximation) *Let* S *be a subset of* $\mathbb{R}$*. The hull approximation of* S*, denoted* $\square S$*, is the smallest interval* $\mathbf{x}$ *such that* $S \subseteq \mathbf{x}$*.*

By definition the hull approximation is unique. In the case of interval arithmetic over floating-point numbers, the hull approximation corresponds to the

[10] Many books use the term **sharp** instead of narrow. In this document both are used without distinction.

smallest representable interval $\mathbf{x}$ such that $S \subseteq \mathbf{x}$.

Let $\mathbb{F}$ denote the available set of floating-point numbers in a machine. Let $x^- \in \mathbb{F}$ be the greatest number lower or equal to x, and let $x^+ \in \mathbb{F}$ be the smallest number greater or equal to x. So, related to the machine numbers, a real number x will be represented[11] by the interval $\mathbf{x}_f = \Box x = [x^-, x^+]$, and a real interval $\mathbf{y} = [\underline{y}, \overline{y}]$ will be represented by the interval $\mathbf{y}_f = \Box y = [\underline{y}^-, \overline{y}^+]$.

On the other hand, if $\mathbf{x}$ and $\mathbf{y}$ are two intervals, the result of $\mathbf{x} + \mathbf{y}$ will be always contained in the interval $\mathbf{z}_f = \Box(\mathbf{x} + \mathbf{y}) = [(\underline{x} + \underline{y})^-, (\overline{x} + \overline{y})^+]$. The four arithmetic operations can be adapted in the same way to be implemented in a machine, and the inclusion property of interval arithmetic always holds.

Example 3.6.2 *Consider the same machine used in the last example for computing the expression $z = sin(y) + ln(x)$ with $x = e$ and $y = \pi$. The result is obtained by computing*

$$\begin{array}{lllll} e = 2.718\ldots & \rightarrow & \mathbf{x} = [2.71, 2.72] & \rightarrow & ln(\mathbf{x}) = [0.99, 1.01] \\ \pi = 3.141\ldots & \rightarrow & \mathbf{y} = [3.14, 3.15] & \rightarrow & sin(\mathbf{y}) = [-0.01, 0.01] \\ \mathbf{z} = [0.98, 1.02] & & & & \end{array}$$

Notice that the computed result includes the actual value $z = 1$. Moreover, the computed interval $\mathbf{z} = [0.98, 1.02]$ will always contain both the actual value and the value computed with floating-point arithmetic.

3.6.1 Software and Libraries

Although it is possible to implement interval arithmetic in a machine for a particular programming language or application, there exist several implementations for the most popular languages available in Internet[12].

Maybe the first among them is INTLIB [112, 113], developed in Fortran 77 and after in Fortran 90. It is a set of subroutines based on BLAS for basic interval operations. It uses a simulated direct rounding which makes it portable.

Another well-known library (maybe the most popular) is PROFIL/BIAS [119, 118]. It is written in C++ and provides an object-oriented interface for computing with intervals, interval vectors and matrices. Many classic functions, constants and special operation are also available.

[11] Notice that the floating-point numbers x^- and x^+ represent the rounding towards negative infinity and positive infinity of the real value x, respectively.

[12] See, for example `http://interval.sourceforge.net/interval/index.html` or the website of Interval Computation `http://www.cs.utep.edu/interval-comp/`, for details.

As a public domain software, it can be downloaded[13] and used in several operating systems and platforms. It is a complete and very fast interval library for C++ users.

Currently, there are some proposals to add interval arithmetic to the C++ Standard Library (see [23], for example), *Boost Interval Arithmetic Library*[14]. and *filib++* (Fast Interval LIRrary[132]) are some examples.

Commercial environment for linear algebra, scientific calculus and mathematics also include packages for computing with intervals (see [116] for `Mathematica`, Intpak [41, 43] and IntpakX [120] for `Maple`, and [101, 102] for `MS Excel`, for example).

3.7 Generalized and Modal Intervals

This section presents a brief introduction to generalized[15] and modal intervals, with a special attention in the semantics of a given expression. It is not considered as a part of classic interval analysis but as another branch called Modal Interval Analysis (MIA) introduced by Gardeñes [64, 66] in the eighties.

Consider, for instance, the following equation

$$\mathbf{x} + \mathbf{y} = \mathbf{z} \tag{3.40}$$

with $\mathbf{x} = [1, 2]$ and $\mathbf{y} = [0, 5]$. It is clear that the value of $\mathbf{z}$ must be the set of values $\{z = x + y \mid x \in \mathbf{x} \text{ and } y \in \mathbf{y}\}$, and that is $\mathbf{z} = [1, 7]$. Now, consider the same equation with the values $\mathbf{y} = [0, 5]$ and $\mathbf{z} = [1, 7]$. What interval $\mathbf{x} \in \mathbb{IR}$ verifies $\mathbf{x} + \mathbf{y} = \mathbf{z}$?. This is an *interval equation*, and the answer to this question cannot be obtained using the expression

$$\mathbf{x} = \mathbf{z} - \mathbf{y} \tag{3.41}$$

as shown in example 3.2.2 in section 3.2. Moreover, the equation $\mathbf{x} + \mathbf{y} = \mathbf{z}$ has a solution for $\mathbf{x} \in \mathbb{IR}$ only when $w(\mathbf{z}) \geq w(\mathbf{y})$, and the solution is $\mathbf{x} = [\underline{z} - \underline{y}, \overline{z} - \overline{y}]$. The last expression is called *algebraic difference* [66] (in opposition to *interval difference* defined in section 3.2).

[13] From `http://www.ti3.tu-harburg.de/knueppel/profil/Profil2.tgz`

[14] See `http://www.boost.org/libs/numeric/interval/doc/interval.htm`

[15] Also called directed interval (see [138]).

Example 3.7.1 *Consider the interval equation* $\mathbf{x} + \mathbf{y} = \mathbf{z}$*, and the list of values for* $\mathbf{y}$ *and* $\mathbf{z}$*. The interval value* $\mathbf{x}$ *that verifies this equation is*

$$\mathbf{y} = [1, 2],\ \mathbf{z} = [5, 8] \implies \mathbf{x} = [5 - 1, 8 - 2] = [4, 6] \tag{3.42}$$

$$\mathbf{y} = [0, 9],\ \mathbf{z} = [0, 9] \implies \mathbf{x} = [0 - 0, 9 - 9] = [0, 0] \tag{3.43}$$

$$\mathbf{y} = [2, 8],\ \mathbf{z} = [3, 6] \implies \mathbf{x} = [3 - 2, 6 - 8] = [1, -2] \tag{3.44}$$

Notice that only equation (3.42) and equation (3.43) have a solution $\mathbf{x} \in \mathbb{IR}$. The solution of equation (3.44) is not an interval (in the sense of the definition 3.3 in section 3.1). Notice that as $w(\mathbf{z}) < w(\mathbf{y})$, there exists no value $\mathbf{x} \in \mathbb{IR}$ which verifies the equation $\mathbf{x} + [2, 8] = [3, 6]$.

The result $\mathbf{x} = [1, -2]$ (an improper interval) of equation (3.44) is an object of another set called *generalized interval*, and introduced by Kaucher [56, 57, 111] in his *Kaucher arithmetic*. This new set of numbers coupled with Kaucher arithmetic extends the classic interval analysis allowing new interpretations to interval expressions.

3.7.1 Generalized Intervals

As its classic counterpart, a generalized interval $\mathbf{x} = [\underline{x}, \overline{x}]$ is defined by two real numbers $\underline{x}, \overline{x} \in \mathbb{R}$, but without imposing the constraint $\underline{x} \leq \overline{x}$. The set of generalized intervals is denoted by $\mathbb{KR}$ and it is related with the set of classic intervals by the relation

$$\mathbb{KR} = \mathbb{IR} \cup \overline{\mathbb{IR}}$$

where $\mathbb{IR}$ is the set of classic intervals (with $\underline{x} \leq \overline{x}$), and $\overline{\mathbb{IR}}$ is the set of *improper* intervals (with $\underline{x} > \overline{x}$). Some of the definitions given for classic intervals are also applied for generalized intervals. Some examples are:

1. Equality:
   $\mathbf{x} = \mathbf{y} \iff (\underline{x} = \underline{y})$ and $(\overline{x} = \overline{y})$
2. Inclusion:
   $\mathbf{x} \subseteq \mathbf{y} \iff (\underline{x} \geq \underline{y})$ and $(\overline{x} \leq \overline{y})$
3. Midpoint:
   $m(\mathbf{x}) = \frac{1}{2}(\underline{x} + \overline{x})$

Some other definitions have been added in order to change the proper/improper quality of a generalized interval keeping unchanged the underlying set of associated

reals.

1. Dual operator:
   $Dual([\underline{x}, \overline{x}]) = [\overline{x}, \underline{x}]$

2. Proper projection:
   $Pro([\underline{x}, \overline{x}]) = [\min\{\underline{x}, \overline{x}\}, \max\{\underline{x}, \overline{x}\}]$

3. Improper projection:
   $Imp([\underline{x}, \overline{x}]) = [\max\{\underline{x}, \overline{x}\}, \min\{\underline{x}, \overline{x}\}]$

Operations *meet* and *join* between two generalized intervals $\mathbf{x}$ and $\mathbf{y}$ (see section 3.3) are extended by relaxing the constraint $max(\underline{x}, \underline{y}) \leq min(\underline{x}, \underline{y})$. Thus

$$\begin{aligned} \mathbf{x} \wedge \mathbf{y} &= [max(\underline{x}, \underline{y}), min(\underline{x}, \underline{y})] \\ \mathbf{x} \vee \mathbf{y} &= [min(\underline{x}, \underline{y}), max(\underline{x}, \underline{y})] \end{aligned}$$

are closed in $\mathbb{KR}$.

3.7.2 Kaucher Arithmetic

In Kaucher interval arithmetic (also called *complete interval arithmetic* [190] or *directed interval arithmetic* [138]) intervals form a commutative group with respect to addition[16] and a complete lattice with respect to inclusion. As its classic counterpart, addition and substraction are defined using the bounds of the intervals as follows:

- Addition
$$\mathbf{x} \oplus \mathbf{y} = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$$

- Substraction
$$\mathbf{x} \ominus \mathbf{y} = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$$

The expression $\mathbf{x} \otimes \mathbf{y}$ for interval multiplication in $\mathbb{KR}$ is more complicated than the classic version. It is shown in table 3.3. The symbol $\mathbb{KR}^+$ represents the set of intervals with positive bounds[17], that is $\{\mathbf{x} \in \mathbb{KR} \mid (\underline{x} > 0) \text{ and } (\overline{x} > 0)\}$, while the symbol $\mathbb{KR}^-$ represents the set of intervals with negative bounds $\{\mathbf{x} \in \mathbb{KR} \mid (\underline{x} < 0) \text{ and } (\overline{x} < 0)\}$.

[16] Intervals together with addition operation satisfy the four fundamental properties of *closure*, *associativity*, the *identity property*, and the *inverse property*.

[17] Also called *positive intervals*.

	$\mathbf{y} \in \mathbb{KR}^+$	$0 \in \mathbf{y}$	$\mathbf{y} \in \mathbb{KR}^-$	$\mathbf{y} \subseteq 0$
$\mathbf{x} \in \mathbb{KR}^+$	$[\underline{xy}, \overline{xy}]$	$[\overline{x}\underline{y}, \overline{xy}]$	$[\overline{x}\underline{y}, \underline{x}\overline{y}]$	$[\underline{xy}, \underline{x}\overline{y}]$
$0 \in \mathbf{x}$	$[\underline{x}\overline{y}, \overline{xy}]$	$[min\{\underline{x}\overline{y}, \overline{x}\underline{y}\}, max\{\underline{xy}, \overline{xy}\}]$	$[\overline{x}\underline{y}, \underline{xy}]$	0
$\mathbf{x} \in \mathbb{KR}^-$	$[\underline{x}\overline{y}, \overline{x}\underline{y}]$	$[\underline{x}\overline{y}, \underline{xy}]$	$[\overline{xy}, \underline{x}\overline{y}]$	$[\overline{xy}, \overline{x}\underline{y}]$
$\mathbf{x} \subseteq 0$	$[\underline{xy}, \overline{x}\underline{y}]$	0	$[\overline{xy}, \underline{x}\overline{y}]$	$[max\{\underline{xy}, \overline{xy}\}, min\{\underline{x}\overline{y}, \overline{x}\underline{y}\}]$

Table 3.3: Multiplication in complete interval arithmetic.

Interval division is defined as its classic counterpart

$$\mathbf{x} \oslash \mathbf{y} = \mathbf{x} \otimes \mathbf{y}^{-1} = \mathbf{x} \otimes [1/\overline{y}, 1/\underline{y}] \qquad \text{for } \underline{y}\overline{y} > 0$$

Notice that division by an interval $\mathbf{x} \ni 0$ or $\mathbf{x} \subseteq 0$ is forbidden. Anyway, the most important property of classic interval arithmetic (inclusion monotony) holds in complete interval arithmetic too. That is

$$\mathbf{x} \subseteq \mathbf{y} \;\text{ and }\; \mathbf{z} \subseteq \mathbf{w} \;\Rightarrow\; \mathbf{x} \diamond \mathbf{z} \;\subseteq\; \mathbf{y} \diamond \mathbf{w}$$

for $\diamond \in \{\oplus, \ominus, \otimes, \oslash\}$ and $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w} \in \mathbb{KR}$.

Example 3.7.2 *Remember the interval equation* $\mathbf{x} + \mathbf{y} = \mathbf{z}$ *presented in example 3.7.1, with* $\mathbf{y} = [2, 4]$ *and* $\mathbf{z} = [3, 8]$. *The solution* $\mathbf{x} \in \mathbb{KR}$ *that verifies this equation can be easily obtained using complete interval arithmetic and the* Dual *operator as follows*

$$\begin{array}{lll} \mathbf{x} + \mathbf{y} = \mathbf{z} & \Longrightarrow & \mathbf{x} + \mathbf{y} - Dual(\mathbf{y}) = \mathbf{z} - Dual(\mathbf{y}) \\ \mathbf{x} + [0, 0] = \mathbf{z} - Dual(\mathbf{y}) & \Longrightarrow & \mathbf{x} = [3, 8] - [4, 2] \\ \mathbf{x} = [1, 4] & & \end{array}$$

Notice that $\mathbf{x} = [1, 4]$ *is the solution of the equation in classic interval arithmetic (because* $\mathbf{x} \in \mathbb{IR}$*).*

Another example using $\mathbf{y} = [2, 5]$ and $\mathbf{z} = [3, 4]$ shows that the solution $\mathbf{x} = [1, -1]$ is not an interval in the classic sense ($\mathbf{x} \notin \mathbb{IR}$), so the equation has no

solution in classic intervals arithmetic. The remaining question is, *What is the sense of a solution* $\mathbf{x} = [1, -1]$*?* The answer has been pointed out by Gardeñes and studied by several authors (see [126, 190]).

3.7.3 Modal Intervals

Modal interval analysis (MIA) is a special tool introduced by Gardeñes (see [64, 65, 66]) which studies the semantics behind the operations between intervals. May be the most important results from MIA are the relationship between *proper/improper* intervals as sets of uncertain/controlable values (with an associated quantifier), and the *interpretability* of the result of an operation between them.

Example 3.7.3 *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *represent the uncertain (but limited) length of two cords. If* $\mathbf{x} = [5, 7]$ *and* $\mathbf{y} = [8, 10]$*, can I make a new cord with a length 15 using these cords? The answer is clearly* you cannot! *Why?, because you do not know the length of the cords. The only available information is that the lengths of cords* $\mathbf{x}$ *and* $\mathbf{y}$ *are bounded, so the result of the operation*

$$\mathbf{x} + \mathbf{y} = [5, 7] + [8, 10] = [13, 17]$$

gives the uncertain bounded *length of a resulted cord build from* $\mathbf{x}$ *and* $\mathbf{y}$*.*

In classic interval arithmetic, the only semantic interpretation of an operation $\diamond$ between two intervals $\mathbf{x}$ and $\mathbf{y}$ is as follows

$$(\forall x \in \mathbf{x})(\forall y \in \mathbf{y})(\exists z \in \mathbf{z})(x \diamond y = z)$$

So, in the last example, the only information we have is that there exists a value $z \in [13, 17]$ but we do not know it. This is a limitation of classic interval arithmetic. Although we can control the value of a variable (for example, cutting the first cord in a desired long between 5 and 7), we cannot express it with classic interval arithmetic.

MIA extends the semantic interpretation of classic intervals by associating a quantifier to each interval. Thus, a modal interval $(\mathbf{x}, Q\mathbf{x})$ is a couple built from a classic interval $\mathbf{x} \in \mathbb{IR}$ and its associated quantifier $Q\mathbf{x} \in \{\exists, \forall\}$. The interval $\mathbf{x}$ is called the *extension* and the quantifier $Q\mathbf{x}$ is the *modality*.

Because this theory has a totally different construction from classic interval analysis, its presentation needs the introduction of many concepts and of course

a special notation. Instead of including the complete theory in this document, a simpler re-formulation proposed by Goldsztejn [71] and based on the practical aspect of modal intervals will be used. The formal definitions of the original theory can be found in [64, 65, 66] and [85].

Classic interval arithmetic is based on the possibility of building interval extension of a given function f, keeping the monotony property. That means to find an interval value $\mathbf{z} \in \mathbb{IR}$ such that $range(f) \subseteq \mathbf{z}$ for a given domain $\mathbf{x}$. On other words, to compute an interval $\mathbf{z} \in \mathbb{IR}$ that verifies

$$(\forall x \in \mathbf{x})(\exists z \in \mathbf{z})(z = f(x))$$

Of course, the sharper the interval $\mathbf{z}$ is, the better the approximation of the range of the function is. Goldsztejn [71] generalizes this problem as following: *Given a function f and set of variables $x_1, \ldots, x_n$ with interval domains $\mathbf{x}_1, \ldots, \mathbf{x}_n$ and an associated quantifier Q_k for each of them, determine a quantifier Q and an interval $\mathbf{z} \in \mathbb{IR}$ such that*

$$(\forall x_A \in \mathbf{x}_A)(Qz \in \mathbf{z})(\exists x_E \in \mathbf{x}_E)(z = f(x)) \tag{3.45}$$

where $A = \{k \mid Q_{x_k} = \forall\}$ and $E = \{k \mid Q_{x_k} = \exists\}$. In other words, to determine an interval result with associated information. Only the case of universal quantifiers preceding the existential ones is dealt and the solution set corresponding to this particular case is called AE-Solution set.

Generalized intervals are used in order to represent domains and quantifiers as follows: the set of real numbers associated to a variable is represented by the proper projection of a generalized interval, while the associated quantifier is represented by its proper/improper character. The convention used for quantifiers is as follows:

- if $\mathbf{x}_k$ is proper then $Q_{x_k} = \forall$ else $Q_{x_k} = \exists$.
- if $\mathbf{z}_k$ is proper then $Q_{z_k} = \exists$ else $Q_{x_k} = \forall$.

Notice that this convention is compatible with classic interval arithmetic because we obtain the same semantic interpretation when only proper intervals are involved.

According to this representation, the problem presented in (3.45) is formulated as follows:

$$(\forall x_P \in \mathbf{x}_P)(Qz \in \mathbf{z})(\exists x_I \in Pro(\mathbf{x}_I))(z = f(x)) \tag{3.46}$$

where the subindices P represent the proper intervals, while I represent the improper ones.

Example 3.7.4 *Let* $\mathbf{x} = [1, 3]$ *and* $\mathbf{y} = [2, 5]$ *represent two generalized interval, and* $f(x, y) = x + y$ *a real function. It is easy to verify that the value* $z = [3, 8]$ *is a solution for (3.45), and the classic semantic interpretation*

$$(\forall x \in [1, 3])(\forall y \in [2, 5])(\exists z \in [3, 8])(z = x + y)$$

holds.

Such interval $\mathbf{z} \in \mathbb{KR}$ is called *interpretable* with respect to f and x (or simply $(f, \mathbf{x})$-interpretable). Actually, there exist several $(f, \mathbf{x})$-interpretable intervals for a given problem, but an interval z_i will be better (more accurate) than an interval z_j if $z_i \subseteq z_j$ (generalized interval inclusion).

The best[18] interval $\mathbf{z} \in \mathbb{KR}$ (in the sense of interval inclusion) for the four basic arithmetic operators $\{+, -, \times, \div\}$ can be easily computed using Kaucher arithmetic. Elementary function (like sqrt, log, exp, etc...) can also be computed using the proper projection of the generalized interval and keeping its proper/improper characteristic in the result.

Example 3.7.5 *Let* $\mathbf{x} = [1, 3]$ *and* $\mathbf{y} = [9, 4]$ *represent two generalized intervals, and* $f_1(x, y) = x + y$, $f_2(y) = \sqrt{y}$. *The solution* $\mathbf{z}$ *for each evaluation (using generalized interval) and their semantic interpretation are computed as follows:*

$$\begin{aligned} \mathbf{z}_1 = \mathbf{f}_1(\mathbf{x}, \mathbf{y}) &\Rightarrow \mathbf{z}_1 = [1, 3] + [9, 4] = [10, 7] \\ &\Rightarrow (\forall x \in [1, 3])(\forall z_1 \in [7, 10])(\exists y \in [4, 9])(z_1 = x + y) \end{aligned}$$

$$\begin{aligned} \mathbf{z}_2 = \mathbf{f}_2(\mathbf{y}) &\Rightarrow \mathbf{z}_2 = Imp(\sqrt{Pro([9, 4])}) = Imp([2, 3]) = [3, 2] \\ &\Rightarrow (\forall z_2 \in [2, 3])(\exists y \in [4, 9])(z_2 = \sqrt{y}) \end{aligned}$$

Notice that the same idea can be applied in order to compute interval evaluation and semantic interpretation for more complicated functions. Using a tree decomposition of the whole expression into simpler binary operations and elementary functions it is possible to compute a *natural* generalized interval evaluation (as computed in classic interval arithmetic for rational expressions) and its semantic interpretation.

[18] It means to compute the smallest proper interval (if $\mathbf{z}$ is proper) or the largest improper interval (if $\mathbf{z}$ is improper).

Example 3.7.6 *Let* $\mathbf{x} = [1, 3]$, $\mathbf{y} = [-1, 2]$, *and* $\mathbf{w} = [6, 0]$ *be three generalized intervals and* $f(x, y, w) = x^2 + 2\sqrt{w - y}$. *The solution* $\mathbf{z}$ *and the associated semantic interpretation can be computed as follows:*

$$\begin{array}{lcl} \mathbf{z}_1 = \mathbf{w} - \mathbf{y} & \Rightarrow & \mathbf{z}_1 = [6, 0] - [-1, 2] = [4, 1] \\ \mathbf{z}_2 = \sqrt{\mathbf{z}_1} & \Rightarrow & \mathbf{z}_2 = Imp(\sqrt{Pro([4, 1])}) = [2, 1] \\ \mathbf{z}_3 = 2 \times \mathbf{z}_2 & \Rightarrow & \mathbf{z}_3 = 2 \times [2, 1] = [4, 2] \\ \mathbf{z}_4 = \mathbf{x}^2 & \Rightarrow & \mathbf{z}_4 = [1, 3]^2 = [1, 9] \\ \mathbf{z} = \mathbf{z}_4 + \mathbf{z}_3 & \Rightarrow & \mathbf{z} = [1, 9] + [4, 2] = [5, 11] \end{array}$$

Based on the final result $\mathbf{z} = [5, 11]$, *the semantic interpretation of the generalized interval evaluation of* f *is*

$$(\forall x \in [1, 3])(\forall y \in [-1, 2])(\exists z \in [5, 11])(\exists w \in [0, 6])(z = x^2 + 2\sqrt{w - y})$$

Nevertheless, the natural interval evaluation of an given expression does not give the best result in a general case. In classic interval arithmetic, for example, when multi-ocurrence variables are involved in the evaluation of a function the result is generally an over-estimation of the range of the function (remember the expression $f(x) = x - x$ with $x = [0, 1]$, the result of the natural interval evaluation is $f([0, 1]) = [-1, 1]$, but $f(x) = 0$ for all x in the interval $[0, 1]$).

The same problem is found when multi-ocurrence variables are involved in a function f over generalized intervals. When multi-ocurrence variables are proper, the computed interval $\mathbf{z}'$ obtained by evaluating f using Kaucher arithmetic is an over approximation of the optimal $\mathbf{z}$ (that means $\mathbf{z} \subseteq \mathbf{z}'$). Otherwise, when multi-ocurrence variables are improper, the computed interval $\mathbf{z}'$ obtained by evaluating f using Kaucher arithmetic is not an interpretable interval. For example, consider the interval function $f(x) = x - x$ with $x = [1, 0]$. The natural interval evaluation given by Kaucher arithmetic is $f([1, 0]) = [1, -1]$, but the value $[1, -1]$ is not a $(f, \mathbf{x})$-interpretable result.

In some cases, a mean-value extension to generalized intervals can be used in order to compute interpretable intervals, but computing the best interpretable interval in a general case is still an open problem.

A better description of the limitation of this approach and some condition for obtaining interpretable intervals for a given function (and for vectorial functions) can be found in [71].

3.8 Conclusions

Computers work with digital numbers, not with real ones. This is a very important fact that must be taken into account when a set of operations is performed by a machine. At the beginnings of interval arithmetic it was a good motivation for developing special tools to handle rounding errors and floating-point arithmetic limitations. But the whole Interval Analysis theory is much more powerful and not only protect the correctness of results against rounding errors, but provides elaborated algorithms for solving systems of linear and non-linear equations in a certified[19] way, among other things.

Many of these tools have been successfully used in Constraint Programming over continuous domains (generally represented by intervals). The combination of both Constraint Programming and Interval Analysis has shown to improve the performance in problem solving and has been used in many other areas.

In this chapter, a brief introduction of Interval Analysis and their tools has been presented. The combination of these and the Constraint Programming tools is given in the next chapter. The contributions of this thesis are based on this combination.

[19]The expression *certified way* means that no solution is lost during the system solving process.

Chapter 4

Constraint Programming and Intervals

Although Constraint Programming and Interval Analysis may seem two very different and distant approaches for problem solving (from the *origins and objectives* point of view), they have many things in common.

The inclusion of continuous domains in the area of Constraint Programming brings the natural use of intervals and their associated arithmetic. This fact not only brought the knowledge of Interval Analysis to the set of tools already developed in Constraint Programming but also allowed the development of new tools used afterward by both communities.

The aim of this chapter is to present the combination of Interval Analysis and Constraint Programming for solving CSPs over continuous domains and a set of tools generated by this combination. These tools are the basis of the second part of the thesis which contains the contributions.

Section 4.1 presents an introduction of the evolution of Constraint Programming over continuous domains. Section 4.2 introduces the concept of Numeric Constraint Satisfaction Problem (NCSP) as a special class of CSP. Section 4.3 presents some of the most popular consistency techniques over continuous domains, while Section 4.4 presents different domain splitting strategies. Section 4.5 discusses the problem of the uncertainty in the input data (called parameters), while Section 4.7 introduces some special techniques for solving systems of distance equations. Finally, Section 4.8 presents the conclusions of the chapter.

4.1 Introduction

Among the first works combining Intervals and Constraint Programming we can cite Cleary [33] (who proposed an interval representation to handle operations over de reals in Prolog, and included a propagation algorithm to be used with intervals), Davis [48] (who introduced some theoretical results about the complexity of such a propagation), and Hyvönen [100] (who addressed problems with inexact input/output values and under-constrained situations, and used intervals for propagating simultaneous alternative values, which he called *Tolerance Propagation*). Some of these ideas were after developed in a more formal way as consistency techniques over continuous domains. Examples of those are the 2B-consistency [133] (also known as Hull-consistency [13]), 3B-consistency [133], Box-consistency [16], and Bound-consistency over continuous domains.

Another advantage of using intervals is the guarantee. When intervals are used to guarantee results, the inexactitude of the input data is reflected in the solutions of the problem. Therefore, the solutions of a NCSP involving inexact input data are no more isolated points but a continuum of feasible points in the space (this is also the case of under-constrained NCSP). That brings an additional difficulty which is the *representation* of constraints and their solutions. A special representation of constraints has been introduced in [186].

4.2 Numeric Constraint Satisfaction Problem

A system of equations is basically a CSP over continuous domains (also called a Numeric Constraint Satisfaction Problem or simply NCSP). The definition of a NCSP is almost the same as for a CSP (see section 2.2) except that variable domains are defined by intervals[1] and the constraints are defined by any numeric relation linking a set of variables.

NCSPs are very common in the real life and have been widely studied in many areas (generally posed as system of equations). In this document, a NCSP is considered as a system of equations and/or inequalities (possibly with interval parameters).

Example 4.2.1 *Consider the following NCSP:*

$P = (X, D, C)$

[1] A more general definition considers variable domains as sets of numbers without forcing the use of intervals. In this document only interval domains will be considered.

$X = \{x, y\}$
$D = \{D_x = [0, 5], D_y = [0, 5]\}$
$C = \{c_1 : x^2 + y^2 = 4, \quad c_2 : (x-3)^2 + (y-2)^2 = 4\}$

Figure 4.1 graphically shows the solutions of the system.

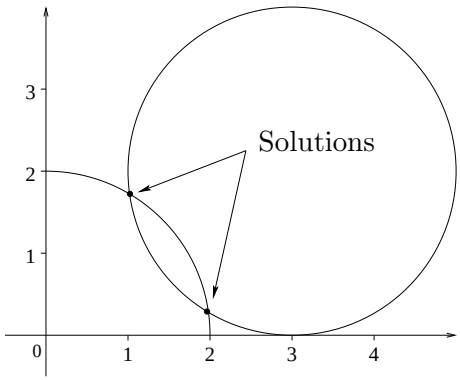


Figure 4.1: Example of a NCSP and its solutions.

In the last example, the real (fixed) values are parameters of the problem. Most of the time they are considered as simple real values but in some cases they can be intervals with a associated quantifier (see Section 4.5.1).

4.3 Filtering Techniques

Roughly speaking, a filtering technique is a method to eliminate values from the domains of the variables that cannot belong to any solution of the problem. This is the case, for example, of the consistency techniques for CSP over finite domains (discussed in Section 2.3.3) and the contracting operators (discussed in Section 3.5.2).

There exist two types of filtering techniques:

- *Local filtering technique:* which considers only one constraint at a time, and propagates the domain reductions due to this constraint through the whole system.
- *Global filtering technique:* which considers all the constraints simultaneously.

Section 3.5.2 discussed some global filtering techniques (as the interval Newton and Krawczyk operators). This section presents four of the most popular local filtering techniques known as *consistency techniques over continuous domains*. Some of them are extensions of those presented in Section 2.3.3 and others are combinations of Numerical Analysis (Interval Analysis) and Constraint Programming.

The process of enforcing a consistency technique is known as *filtering* or *narrowing*. As the same as in finite domains, the filtering phase is performed by propagation, reducing the domain of a given variable (based on a given constraint) and propagating this reduction along the network. Algorithm 5 presents the basis of the propagation.

Algorithm 5 Propagation($P = (X, D, C)$)

```
Q ← {⟨x_i, c_j⟩ | x_i ∈ Vars(c_j) ∧ c_j ∈ C}
while Q not empty do
   select and delete ⟨x_i, c_j⟩ from Q
   if narrowing(x_i, c_j) then
      Q ← Q ∪{⟨x_k, c⟩ | (c ≠ c_j) ∧ (x_i, x_k ∈ Vars(c))}
return
```

The algorithm uses a queue of pairs variable-constraint to be revised. The *narrowing()* operator represents a consistency technique which is enforced on the variable x_i for a given constraint c_j. If the domain of the variable x_i is reduced, the set of variables related to x_i through a constraint (and the respective constraint) is added to the queue. The algorithm finishes when no reduction is possible and the queue is empty.

The main difference between propagation algorithms is in the consistency technique applied in the narrowing operator. Among the most popular techniques are 2B-consistency, 3B-consistency, Box-consistency and Bound-consistency[2]. They are briefly explained below. More information about them and techniques for improving their performance can be obtained in [130].

4.3.1 2B-consistency

The 2B-consistency can be considered as a generalization of the arc-consistency technique (see Section 2.3.3) originally proposed for CSP over finite domains. It

[2]There exists a difference between the Bound-consistency in finite domains (presented in Section 2.3.3) and the Bound-consistency over continuous domains which is an adaptation of the 3B-consistency.

was introduced in [133] and after presented under the name *Hull-consistency* [13]. It is based on an important concept in NCSP which is the *projection* of a constraint on a given variable.

Definition 10 (Projection of a Constraint) *Let $x_1, \ldots, x_k$ be a set of variables and $S = D_1 \times \ldots \times D_k$ a subspace. The projection on x_i of a constraint $c(x_1, \ldots, x_k)$ restricted to domains $D_1, \ldots, D_k$ (denoted by $\Pi_i(c, S)$) is the set: $\{v_i \in D_i \mid \exists (v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k) \in D_1 \times \ldots \times D_{i-1} \times D_{i+1} \times \ldots \times D_k$ such that $c(v_1, \ldots, v_n)$ is verified$\}$.*

Example 4.3.1 *Consider the constraint $c_2 : (x-3)^2 + (y-2)^2 \leq 4$ presented in the example 4.2.1. The projection of c_2 on the variable x is $\Pi_x(c_2, D_x \times D_y) = [1, 5]$ while the projection on y is $\Pi_y(c_2, D_x \times D_y) = [0, 4]$.*

Note that the projection of a constraint on a given variable is not necessarily an interval. Actually, it may be an union of intervals which is commonly approximated by a single interval (including all of them) for practical issues.

Definition 11 (2B-consistency) *Let P be a NCSP and x a variable of P with domain $D_x = [a, b]$. D_x is 2B-consistent iff $\forall c(x, x_1, \ldots, x_k)$ a constraint over x:*

- *$\exists v_1, \ldots, v_k \in D_1 \times \ldots \times D_k \mid c(a, v_1, \ldots, v_k)$ is satisfied,*
- *$\exists v_1, \ldots, v_k \in D_1 \times \ldots \times D_k \mid c(b, v_1, \ldots, v_k)$ is satisfied.*

A NCSP is 2B-consistent if all its domains are 2B-consistent.

As shown in the definition, the 2B-consistency is restricted only to the bounds of the variable domains. That means some values of a domain may be locally inconsistent[3]. That is similar to the Bound-consistency over finite domains presented in Section 2.3.3.

Definition 12 (Closure by 2B-consistency) *Given a NCSP $P = (X, D, C)$, a closure by 2B-consistency of P (denoted $\Phi_{2B}(P)$), is a NCSP $P' = (X, D', C)$ such that:*

- *P' is equivalent to P,*
- *P' is 2B-consistent,*

[3]Actually, when the projection of a constraint on a variable is a set of disjoint intervals, each gap between these intervals contains inconsistent values.

- *D' is the biggest domain $D' \subseteq D$ such that P' is 2B-consistent.*

The closure $\Phi_{2B}(P)$ always exists and is unique.

A closure by 2B-consistency can be implemented in several ways. The basic algorithm for enforcing 2B-consistency is based on the projection of each constraint on each variable (and looks like that presented in Section 2.3.3 for arc-consistency). Other implementations use symbolic manipulations of the constraints in order to obtain better interval evaluations (see [142], for an example).

In the original presentation, the algorithm was designed in terms of simple primitive constraints, and the system should be decomposed into primitives.

An important improvement to the original algorithm was introduced in [15] under the name HC4 (algorithm for enforcing Hull-consistency). It removes the decomposition of a constraint into primitives and enforces hull-consistency over complex constraints. The main idea is to use the syntactic tree of a constraint c and to perform a forward evaluation (starting from the leaves to the root of the tree) and a backward evaluation (from the root to each leaf representing a variable) in order to project the whole constraint on each variable. At the root of the tree, both child nodes are intersected and the backward propagation begins. Figure 4.2 shows an example of the narrowing operator of HC4 (called HC4Revise) using both forward and backward propagations to compute the projection of $c : y = z - x^2$ on the variables x, y, and z.

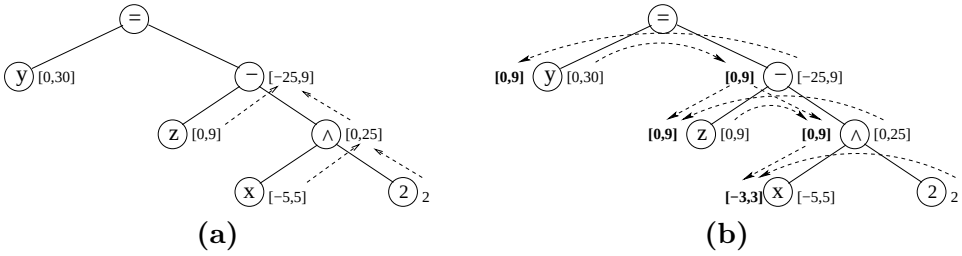


Figure 4.2: Example of the forward (a) and backward (b) propagations performed by the algorithm HC4Revise for computing the projection of $c : y = z - x^2$ with initial domain $\mathbf{x} = [-5, 5]$, $\mathbf{y} = [0, 30]$, and $\mathbf{z} = [0, 9]$ on each variable.

Lhomme [133] identified a termination problem of the algorithm for enforcing 2B-consistency, which is shown in the following example:

Example 4.3.2 *Consider the following NCSP:*

$P = (X, D, C)$

$$X = \{x, y\}$$
$$D = \{D_x = [0, 10], D_y = [0, 100]\}$$
$$C = \{c_1 : y = x + 1, \quad c_2 : y = 2 * x\}$$

Considering $D_x = [0, 10]$ *the projection of* c_1 *on* y *leads to* $D_y = [1, 11]$. *Using this new domain for* y *and the constraint* c_2 *we obtain a projection* $D_x = [0.5, 5.5]$. *Actually, this process continues reducing alternately* D_x *and* D_y *with an asymptotic convergence towards the solution* $x = 1, y = 2$.

A possible solution for this problem is the inclusion of a *precision coefficient* w. The resulting consistency is called 2B(w)-consistency, and the main idea is to revise again the constraints containing a variable x only if the domain of x was *sufficiently* reduced in the last projection.

It is important to note that the closure $\Phi_{2B(w)}(P)$ is not unique. It depends on the order in which constraints are evaluated (only $\Phi_{2B(0)}(P)$ is unique because it is equivalent to the 2B-consistency). Anyway, the 2B(w)-consistency is the most frequently implemented in practice.

4.3.2 3B-consistency

It is important to note that enforcing a 2B-consistency for a given problem does not always reduce the domain of the variables to the smallest interval containing the projection of the solutions of the problem.

Example 4.3.3 *Consider the following NCSP:*

$$P = (X, D, C)$$
$$X = \{x, y\}$$
$$D = \{D_x = [0, 5], D_y = [0, 5]\}$$
$$C = \{c_1 : (x-1)^2 + (y-1)^2 \leq 1, \quad c_2 : (x-2)^2 + (y-1)^2 \leq 1\}$$

Figure 4.3 shows the resulting box after enforcing 2B-consistency on the problem P. *The final domains* $D_x = [1, 2]$ *and* $D_y = [0, 2]$ *are 2B-consistent, but even though the value* $2 \in D_y$ *satisfies both constraints (separately), it does not belong to any solution of the problem.*

Lhomme [133] introduced a stronger consistency called 3B-consistency which allows one to obtain a better rough approximation of the solutions of the problem. It can be seen as a form of strong 3-consistency [60] restricted to the bounds of the domains (see Section 2.3.3).

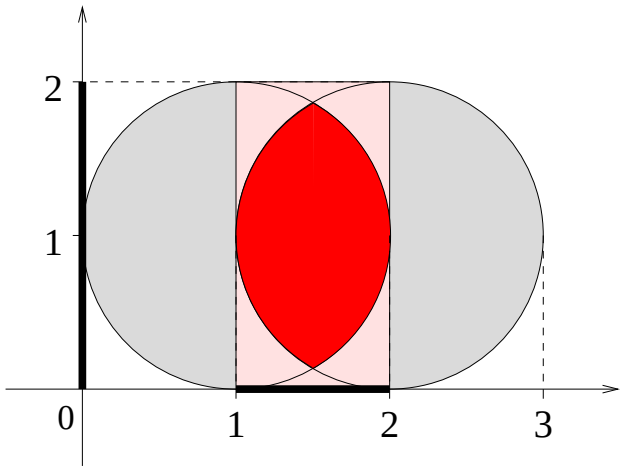


Figure 4.3: Example of a 2B-consistent problem.

Definition 13 (3B-consistency) *Let P be a NCSP and x a variable of P with domain $D_x = [a, b]$. The domain D_x is 3B-consistent iff*

- $\Phi_{2B}(P')$ *is not empty, where* P' *is derived from* P *by replacing* $D_x = [a, a^+)$,
- $\Phi_{2B}(P'')$ *is not empty, where* P'' *is derived from* P *by replacing* $D_x = (b^-, b]$.

A NCSP is 3B-consistent if all its domains are 3B-consistent.

In other words, a domain is 3B-consistent if it can be reduced to one of its bounds and enforce a 2B-consistency over the resulting NCSP (without producing an empty set). The closure of a NCSP P by 3B-consistency is denoted by $\Phi_{3B}(P)$ and can be theoretically obtained by removing the part of the domain which is not consistent.

In practice, the algorithm commonly implemented enforces another consistency called 3B(w_1, w_2)-consistency which follows the same idea as the 2B(w)-consistency presented in Section 4.3.1. The 3B(w_1, w_2)-consistency tries to avoid the asymptotic convergence by considering a form of *precision coefficient* for the 2B-consistency and the 3B-consistency processes.

Definition 14 (3B(w_1, w_2)-consistency) *Let P be a NCSP, x a variable of P with domain $D_x = [a, b]$, and $w_1 \geq w_2 \geq 0$. D_x is 3B(w_1, w_2)-consistent iff*

- $\Phi_{2B(w_2)}(P')$ *is not empty, when* D_x *is replaced by* $[a, a^{+w_1}]$ *in* P',
- $\Phi_{2B(w_2)}(P'')$ *is not empty, when* D_x *is replaced by* $[b^{-w_1}, b]$ *in* P''.

A NCSP is 3B(w_1, w_2)-consistent if all its domains are 3B(w_1, w_2)-consistent.

In some implementations the coefficient w is considered as a fixed precision value and if the projection of a constraint c produces a domain reduction less than w (on a variable x) then this reduction is not more propagated to the other constraints (that means the set of constraints containing the variable x are not revised again).

4.3.3 Box-consistency

Box-consistency was first introduced in [16] as an approximation of arc-consistency and whose implementation uses the Newton interval method. It avoids the decomposition of a constraint into primitives (tackling the dependency problem of intervals for variables with many occurrences). Roughly speaking, the idea is to compute the left-most and right-most zeros of an univariate interval function $\mathbf{f}_x$ obtained by replacing all variables but one (x) by their interval domains.

Definition 15 (Box-consistency) *Let $P = (X, D, C)$ be a NCSP and $c \in C$ a n-ary constraint over the variables $x_1, \ldots, x_n$ with interval domains $\mathbf{x}_1, \ldots, \mathbf{x}_n$. The constraint c is box-consistent (w.r.t. $\mathbf{x}_1, \ldots, \mathbf{x}_n$) iff*

$$\mathbf{x}_i = hull(\mathbf{x}_i \cap \{a_i \in \mathbb{R} \mid c(x_1, \ldots, x_{i-1}, \tilde{a}_i, x_{i+1}, \ldots, x_n) \text{ holds}\})$$

A NCSP is Box-consistent if all its constraints are Box-consistent.

The value $\tilde{a}_i$ in the definition denotes the computational safe representation of the real value a_i. In other definitions (see [34]) it is replaced by the open interval $[\underline{a}_i, \underline{a}_i^+)$ (for the lower bound) and $(\overline{a}_i^-, \overline{a}_i]$ (for the upper bound).

The closure by Box-consistency is defined in the same way as it was defined for 2B-consistency and is denoted $\Phi_{Box}(P)$. The algorithm for enforcing Box-consistency follows the generic Algorithm 5 with a narrowing operator which computes the left-most quasi-zero as shown in Algorithm 6 (see [34]) and the right-most quasi-zero in an analogous way.

The function $\mathbf{f}_x$ corresponds to the constraint c expressed in the form $c : \mathbf{f}_x = 0$ when all variables but x are replaced by their interval domain. The interval $[\underline{i}, \underline{i}^+]$ represents the safe approximation of the possible left-most zero of $\mathbf{f}_x$.

It can be proved that the interval obtained by computing the left-most and right-most quasi-zeros of a function $\mathbf{f}_x$ corresponding to a constraint c for a variable x, is equivalent to a simple projection of c on the variable x when x appears

Algorithm 6 LNAR($\mathbf{f}_x, \mathbf{x}$)

```
r ← x̄
if 0 ∉ f_x(x) then
  return ∅
else
  i← NEWTON(f_x, x)
  if 0 ∈ f_x([i, i⁺]) then
    return [i, r]
  else
    SPLIT(i, i_1, i_2)
    l_1 ← LNAR(f_x, i_1)
    if l_1 ≠ ∅ then
      return [l_1, r]
    else
      return [LNAR(f_x, i_2), r]
return
```

exactly once in c. As finding the left-most and right most quasi-zeros is computationally expensive, a new algorithm for enforcing Box-consistency called BC4 was proposed in [15]. This algorithm uses the same narrowing operator that BC3 when the variable occurs more than once in the constraint, and uses HC4revise (the narrowing operator based on projection of the HC4 algorithm) when the variable occurs only once in the constraint.

4.3.4 Bound-consistency over continuous domains

Bound-consistency applies the same principle of the 3B-consistency to the Box-consistency. In other words, Bound-consistency checks whether Box-consistency can be enforced when the domain of a variable is reduced to the value of one of its bounds. It is important to note that this consistency technique is related to continuous domains and based on Box-consistency (even though the name is the same as the Bound-consistency for finite domains defined in Section 2.3.3). Hereafter, this name is related only to the consistency for continuous domains.

Definition 16 (Bound-consistency) *Let P be a NCSP and x a variable of P with domain $D_x = [a, b]$. The domain D_x is Bound-consistent iff*

- $\Phi_{Box}(P')$ *is not empty, where P' is derived from P by replacing $D_x = [a, a^+)$,*
- $\Phi_{Box}(P'')$ *is not empty, where P'' is derived from P by replacing $D_x = (b^-, b]$.*

A NCSP is Bound-consistent if all its domains are Bound-consistent.

Some authors [34] define the Bound-consistency as follows:

Definition 17 (Bound-consistency) *Let P be a NCSP and c a k-ary constraint over the variables $(x_1, \ldots, x_k)$ with interval domains $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$. The constraint c is Bound-consistent iff for all x_i, the following relations hold:*

- $\Phi_{Box}(c(x_1, \ldots, x_{i-1}, [\underline{x}_i, \underline{x}_i^+), x_{i+1}, \ldots, x_k))$ *is not empty,*
- $\Phi_{Box}(c(x_1, \ldots, x_{i-1}, (\overline{x}_i^-, \overline{x}_i], x_{i+1}, \ldots, x_k))$ *is not empty.*

A NCSP is Bound-consistent if all its constraints are Bound-consistent.

In practice, both definitions are equivalent.

4.4 Domain splitting strategies

A *domain splitting strategy* is simply an adaptation of the political proverb *divide ut imperes* (also called **divide-and-conquer** in computer science) that consists in breaking a problem into simpler subproblems of the same type, next to solve these subproblems, and finally to reunify the obtained results into a solution to the original problem. As presented in Section 3.5.1, this is a common strategy used in Interval Analysis for solving systems of non-linear equations. This strategy can be generalized by considering different ways to split the domains. This section explains some of them.

Consider a CSP P with an initial domain D. A domain splitting strategy generates a set of subdomains $D'_1, \ldots, D'_n$ such as $D = D'_1 \cup \ldots \cup D'_n$. The original problem is then broken into a set of problems $P'_1, \ldots, P'_n$ with domain $D'_1, \ldots, D'_n$ respectively. Each problem P'_i is solved separately, and finally the solution of P is obtained as the union of the solutions of the problems P'_i.

In general these subproblems will be easier to solve, either because they have no solution at all or because some techniques for getting the solutions become effective.

Example 4.4.1 *Consider the following CSP:*

$P = (X, D, C)$
$X = \{x\}$

$D = \{[2, 6]\}$
$C = \{c_1 : \frac{sin(x)}{x} = 0\}$

A domain splitting strategy (like bisection in the middle) will split the interval $[2, 6]$ *in two intervals* $D' = [2, 4]$ *and* $D'' = [4, 6]$. *Using Interval Arithmetics we can prove that all values of the expression* $\frac{sin(x)}{x}$ *for* $x \in [4, 6]$ *are into* $[-0.25, -0.04]$, *so the problem with domain* D'' *has no solution. Now we must solve the problem with domain* D'. *The same technique can be applied for splitting this domain in* $[2, 3]$ *and* $[3, 4]$, *and go on.*

Thus it is primarily a recursive method. The common stop condition used for recursivity is a fixed interval width ϵ, called *precision*. The precision constrains the width of the largest interval in D. If all variable domains have a width lower than ϵ, the problem is considered solved, and the solution is the current domain. Other techniques are used to prove uniqueness and/or existence of solutions in the returned domain (see Section 3.5.3 for more details).

In order to apply a domain splitting strategy, two basic questions need to be answered: *Where to split?* and *How to split?*.

The first question (Where to split?) has many proposals, but the most common method is the single splitting mode (i.e. split only one variable at a time).

Among the strategies for determining the variable to be split, the *Round Robin* method is one of the simplest, and does not require additional information from the problem. In this method, the domains of the variables are processed alternatively, so all variable domains are expected to be split (unless their width is lower than the precision).

Other strategies use syntactic or semantic information. For example, the *Large First* strategy consists in selecting first the domain of maximal width, and in some situations this strategy allows one to eliminate large domain spaces without solutions, improving the performance of the solving process.

Another example is the *Maximal Smear function* [115, 87] strategy, which uses a special expression to determinate the variable with the most important impact in the problem (that is, the variable in which f has the strongest slope). Let $\mathbf{J}(x)$ be the interval extension of the Jacobian matrix of the system, and let $x_1, \ldots, x_n$ be the set of variables. The smear function of the variable x_j is defined as follows:

$$s_j = \max_{1 \leq i \leq m} \{|\underline{J_{i,j}}|, |\overline{J_{i,j}}|\}(\overline{x_j} - \underline{x_j})$$

where m represents the total number of functions, $[\underline{J_{i,j}}, \overline{J_{i,j}}]$ is the interval in the

(i,j)-th entry of $\mathbf{J}(x)$, and $[\underline{x_j}, \overline{x_j}]$ is the interval domain of the variable x_j. The variable that will be split will be the one having the largest s_j.

The main drawback in the strategies based on syntactic or semantic information is that some variable domains may be never split, whether because their interval widths are very small (w.r.t. the others variables), or because the influence in the evaluation of the Jacobian matrix is very low. Anyway, these strategies can be easily combined in order to avoid this problem.

The second question (How to split?) has many answers depending on the problem and the solving strategy used. The simplest method is a middle point bisection of the domain, that is, given a variable domain $D_{x_j} = \mathbf{x}_j$, the bisection computes the new domains:

$$D'_{x_j} = \left[\underline{x_j}, \frac{\underline{x_j} + \overline{x_j}}{2}\right] \qquad \text{and} \qquad D''_{x_j} = \left[\frac{\underline{x_j} + \overline{x_j}}{2}, \overline{x_j}\right]$$

Many authors suggest the importance of considering *gaps detection* for splitting domains. In [87], the author suggests the use of gaps detected when applying the interval Newton method, for splitting the domain of the variables. In [170], the author discusses the performance of a global optimization method using different splitting strategies based in gaps detection in the interval Newton Gauss-Seidel method.

More recent strategies for splitting domains based in gaps detection during the filtering phases have been suggested in [15], and studying in depth in [30, 10]. Experimental results show that these strategies may significantly improve the performance of a search process in some particular cases.

4.5 Uncertainty and Approximations

Most of the problems presented up to now are based on the assumptions that the exact values of the input quantities are known. Actually, the data often come from measurements, and measurements are never 100% precise. That means the actual value p of each input quantity may differ from its measurement result $\tilde{p}$. Sometimes, the probabilities of different error values $\delta p = p - \tilde{p}$ are known, but in general only an upper bound for the error is available. In this cases, the only information about the (unknown) actual value p is that p belongs to an interval $\mathbf{p} = [\underline{p}, \overline{p}]$.

Consider for example the parallel robot presented in Figure 4.4. It is built

from two legs (l_1 and l_2) attached to fixed points ($p = (0, 0)$ and $q = (3, 0)$).

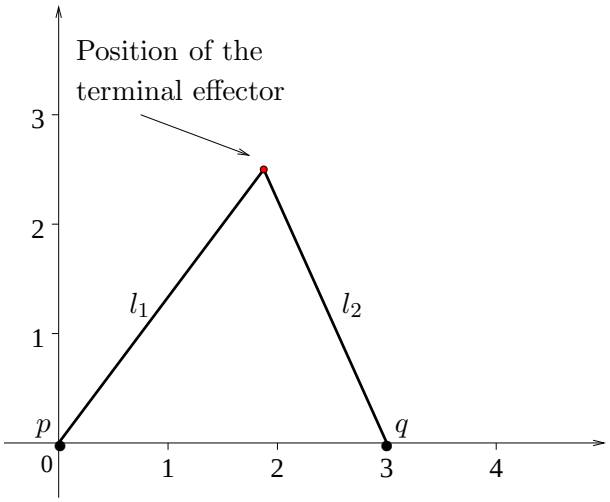


Figure 4.4: Example of a Parallel Robot.

The position of the terminal effector of the robot can be easily computed by solving the following NCSP:

$$
\begin{aligned}
&P = (X, D, C)\\
&\quad X = \{x, y\}\\
&\quad D = \{D_x = [0, 5], D_y = [0, 5]\}\\
&\quad C = \{c_1 : x^2 + y^2 = l_1^2, \quad c_2 : (x-3)^2 + y^2 = l_2^2\}
\end{aligned}
$$

where l_1 and l_2 correspond to the parameters of the system (actually, the positions p and q are also parameters of the system).

If we consider that l_1 and l_2 are measurements and therefore have a degree of *uncertainty*, then a solution point $(\tilde{x}, \tilde{y})$ is nothing more than an approximation of the actual value (x, y). As the only available information is that $l_1 \in \mathbf{l}_1$ and $l_2 \in \mathbf{l}_2$, all values (x, y) for which there exists a value $l_1 \in \mathbf{l}_1$ and $l_2 \in \mathbf{l}_2$ are potential solutions for the problem.

The same problem can also be posed in another way (from the point of view of control). For example, consider an effector placed in each leg (which allows the leg to change its length according to a given interval $\mathbf{l}_i$), and consider that the attachment points are not well known. The question is: *What is the reachable workspace*[4] *of the robot?*

[4]The workspace of a robot corresponds to the set of positions that the terminal effector can

As we are looking for the set of possible positions (x, y) of the terminal effector and we need to be sure that the robot can reach the solution point, we need to compute the following solution set:

$$S = \{s \mid (\forall p \in \mathbf{p})(\forall q \in \mathbf{q})(\exists l_1 \in \mathbf{l}_1)(\exists l_2 \in \mathbf{l}_2)((d(s,p) = l_1) \wedge (d(s,q) = l_2))\}$$

where $d(s,p) = l$ represents a distance constraint between the points s and p. In other words, we are looking for the set of points in the space that we are sure to reach whatever the values p and q are.

We say that this problem has uncertainties or simply that it is a NCSP with uncertainties in its parameters. Sometimes, this types of problems can be transformed into *equivalent* problems in which equations (containing parameters with uncertainty) are replaced by inequalities. For example, the NCSP resulting from replacing a constraint $c : x^2 + y^2 \in \mathbf{l}_1^2$ by $c' : x^2 + y^2 \geq \underline{l_1}^2$ and $c'' : x^2 + y^2 \leq \overline{l_1}^2$ will be equivalent to the original. The advantage here is that the new NCSP has only real valuated parameters (instead of interval ones).

It is clear that these types of problems have no more isolated solutions but one (or several) continuum of solutions. Therefore, classic techniques for problem solving are not adapted to solve them, and new strategies to approximate these continua of solutions must be developed. Section 4.6.1 briefly presents some of them.

4.5.1 Quantified Parameters

As shown in the last section, the parameters of a system can have different meanings (depending on the represented object). For example, measurements of physical quantities (or approximations of them) or range of values which can be controlled by a user. These different meanings are represented by *quantifiers* ($\forall$ and $\exists$).

Constraints involving parameters with an associated quantifier are called quantified constraints. Some examples of quantified constraints are:

$$c_1(x) : (\forall p \in \mathbf{p})(\exists l \in \mathbf{l})((x-p)^2 = l) \tag{4.1}$$

$$c_2(x,y) : (\forall a \in \mathbf{a})(\forall b \in \mathbf{b})(\exists c \in \mathbf{c})((x-a)^2 + (y-b)^2 = c^2) \tag{4.2}$$

$$c_3(z) : (\exists a \in \mathbf{a})(\forall b \in \mathbf{b})(az = z^2 - b) \tag{4.3}$$

reach (i.e. the set of possible poses given a set of parameters).

CSPs involving quantified constraints are called quantified CSPs (in our case QNCSP). There is a special class of QNCSP in which predicates are such that all the universally quantified parameters appear before the existentially quantified ones (as is shown in the constraints (4.1) and (4.2)). The solution sets of this class of problems are called AE-solution sets (see [190] for more details). This is the class of problems treated in this thesis.

4.5.2 Solution Representation

Problems involving inequalities or existentially quantified parameters generally have solution sets with a non-null volume. In other words, the solutions of the problem are not isolated points in the space. In these cases, it is not possible to enumerate all solutions and a better alternative is to describe them in some way.

Sam-Haroud [186] has introduced a special representation of constraints using 2^k-trees, in which nodes are labelled in one of three possible states:

- *white* (or completely legal) which corresponds to boxes containing only points that verify the constraint.
- *gray* (or partially legal) which corresponds to boxes containing both solutions and non-solutions of the constraint.
- *black* (or completely illegal) which corresponds to boxes containing only points that do not verify the constraint.

Figure 4.5 shows an example of the 2^k-tree representation of the constraint $c : x^2 + y^2 \leq 1$. The root node of the tree corresponds to the domain $([-1, 1], [-1, 1])$. A node labelled as white or black never has sub-nodes because all its points are already characterized. Only gray boxes are subdivided (but only until a given precision).

The advantage of this representation is that a conjunction of constraints can be easily computed as a new tree in which the nodes are labelled according to a logic operation between the nodes of the trees representing each constraint.

Two important concepts (based on the label of the box representing the solution space of a problem) are then introduced [185, 186]: the *inner content approximation* of a solution space, given by the white nodes of the tree; and the *outer content approximation* which is given by the union of the white and grey nodes of the tree. Some authors have proposed different definitions for *inner* and *outer* approximations (see [6, 15, 189]).

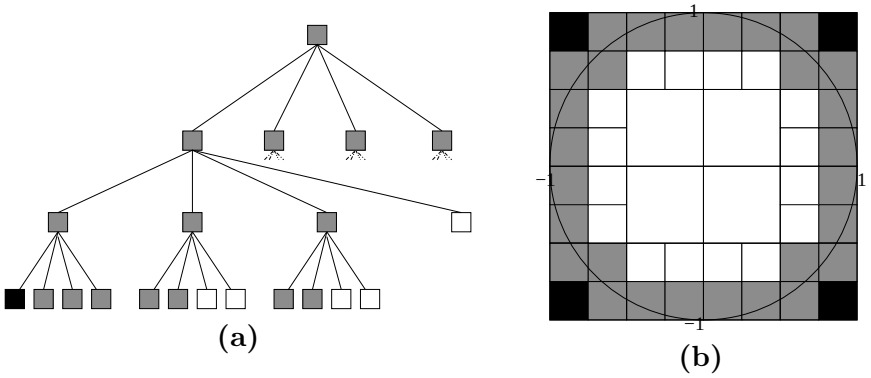


Figure 4.5: (a) A part of the 2^k-tree representation of the constraint $c : x^2+y^2 \leq 1$. (b) The visual interpretation of the tree.

In [6], an inner approximation is basically any box included inside the solution space of an n-ary real relation ρ. In [189], it is a box included in ρ that cannot be extended in any direction without containing a non-solution point, while in [15], the inner approximation contains all the elements included in ρ. This last one is much more similar to the definition proposed in [186], and will be the definition used hereafter in this document[5].

There is a better consensus in the definition of an outer approximation of the solution sets of a problem: it is a conservative enclosure of the set of solutions. Some authors prefer to specify the type of approximation by separating it in *rough outer approximation* (the smallest box containing the set of solutions) and *sharp approximation* which corresponds to an union of boxes describing the set of solutions of the problem. These two types of outer approximations are used in this thesis.

4.6 Solving NCSP

Solving a NCSP is similar to solve a CSP, that means to find an assignment to the variables which verifies all the constraints. Maybe the main difference is that in NCSP we are commonly interested in all the solutions of the problem (instead of only one). Moreover, problems related to robotics need to have more information about the type of solution found and the space around this solution.

[5]From a practical point of view, given a paving of the search space, an inner approximation corresponds to the set of boxes totally included inside the solution space of the problem. This type of approximation is also called *sound approximation*.

Section 4.5 showed that parameters frequently change a punctual solution into a continuum of solutions. This continuum of solutions (or *solution set*) can have different meanings. For example, when the workspace of a parallel robot is computed, this solution set may represent a single configuration[6] disturbed by the influence of the parameters, or to represent several configurations sharing parts of the same space. It is important to know which of them is the case, because if two configurations share the same space then there is at least one singularity in the space in which the robot breaks (see [145] for more details).

Another difficulty of the NCSP with uncertainties is that the set of all solutions cannot be computed in an exact way (actually, even in linear systems computing the exact bounds is NP-hard [125, 178]). Thus the aim is to find a good approximation of the solution set (see Section 4.5.2 for a note about approximations).

4.6.1 The Branch and Prune Algorithm

The branch and prune algorithm is a direct consequence of the combination of an *Evaluation/Bisection* approach (see Section 3.5.1) with consistency techniques (see Section 4.3) and/or contracting operators (see Section 3.5.2). Algorithm 7 shows a generic implementation of a branch and prune algorithm as a solver for NCSP.

Algorithm 7 Solver$(P, \mathbf{x}, w)$

```
Q ← x
Sols ← ∅
while Q not empty do
  select and delete a box x' from Q
  x' ← filtering(P, x')
  if x' ≠ ∅ then
    if width(x') < w then
      Sols ← Sols ∪ x'
    else
      Q ← Q ∪ split_strategy(x')
return Sols
```

The main idea is to generate a set of boxes that conservatively approximate the set of solutions of the problem. The $filtering(P, \mathbf{x}')$ operator corresponds to one (or several) algorithm for enforcing consistencies (as 2B, 3B, Box, or Bound presented in Section 4.3), a contracting operator as interval Newton or Krawczyk

[6] Position of the different parts of the robot that brings the terminal effector to a given point.

(Section 3.5.2), or a combination of both. The *split_strategy*($\mathbf{x}'$) can be a simple bisection in the middle point (as in the original algorithm) or a more elaborated strategy (as presented in Section 4.4).

This algorithm is the kernel of many *state of art* solvers in Constraint Programming over continuous domains. Some examples are ILOG Solver[104], Numerica[94], ALIAS[142], RealPaver[83], and IcosAlias[7]. Obviously, there are many improvements for this generic algorithm and some of them are discussed below.

4.6.2 Improving the Solving Phase

The main drawback of the Branch and Prune algorithm presented in Section 4.6.1 is that the splitting is done blindly. That means the algorithm does not identify more than one type of box: *the box with a width less than a given precision.*

In problems involving inequality constraints (see [127]) or constraints with existentially quantified parameters (see [96]), such a branch and prune algorithm frequently bisect again and again the boxes included inside the solution set, leading to inefficient computations. A simple method to avoid this behavior is to include a test for detecting boxes fully included inside the continuum of solutions.

Interval analysis provides a mechanism for performing such a test (using an interval evaluation of the left-side of the constraint and comparing this evaluation with its right-side).

Another method to perform such a test (for universally quantified constraints involving inequalities) was introduced by Benhamou and Goualard [14]. It is based on the computation of an outer (rough) approximation of the solution set of the negation of the involved constraints.

Example 4.6.1 *Consider the following NCSP:*

$$
\begin{aligned}
&P = (X, D, C)\\
&\quad X = \{x, y\}\\
&\quad D = \{D_x = [0,5], D_y = [0,5]\}\\
&\quad C = \{c : x^2 + y^2 < 4\}
\end{aligned}
$$

and the boxes $\mathbf{b}_1 = ([0,1],[0,1])$, $\mathbf{b}_2 = ([2,3],[2,3])$ *shown in Figure 4.6. The box* $\mathbf{b}_2$ *can be easily removed from the queue of the Algorithm 7 by applying a consistency technique, because no solution of* $c : x^2 + y^2 < 4$ *can be found in* $\mathbf{b}_2$.

[7] IcosAlias is a state of art platform for developing and testing algorithm for solving interval-based problems. It is under development in the COPRIN project (see `http://www-sop.inria.fr/coprin` for more details).

The same cannot be applied to $\mathbf{b}_1$ *because this box contains solutions. So the next step of the algorithm will be the splitting phase.*

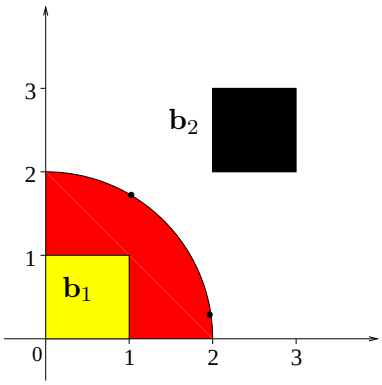


Figure 4.6: Example of internal box ($\mathbf{b}_1$) and external box ($\mathbf{b}_2$).

The original strategy proposed in [14] applies a filtering technique to the negation of the constraint (i.e. $\overline{c} : x^2 + y^2 \geq 4$). As no solution exists for the last constraint in $\mathbf{b}_1$, the filtering technique will return an empty set. This implies that all the points inside $\mathbf{b}_1$ are solutions of the original constraint.

The advantage of this approach is that boxes which are proved to lie inside the solution set will not be bisected any more.

In [191], this approach is used to improve the splitting phase by computing a point which separes the original box in two new boxes: a box containing only solutions of the problem and another box with no particular meaning.

Example 4.6.2 *Consider the same NCSP presented in example 4.6.1, and the box* $\mathbf{b} = ([1,3],[0,1])$ *shown in Figure 4.7. A filtering technique applied to* $\mathbf{b}$ *can reduce it to the new box* $\mathbf{b}' = ([1,2],[0,1])$*. Before applying a split strategy to* $\mathbf{b}'$*, we apply a filtering technique using the negated constraint* $\overline{c}$ *and* $\mathbf{b}'$*. The part of* $\mathbf{b}'$ *removed by the this filtering is proved to contain only solutions of the original problem and then we obtain a good point to perform the splitting phase.*

Both approaches (the *negation test* for detecting inner boxes and the strategy of splitting using the negation test) have been combined with a new representation (called extreme vertex representation EVR) in [204, 205] for solving problems with non-isolated solutions. The preliminary results showed that this combination can improve the performance of both *computing time* and *space requirements* (to

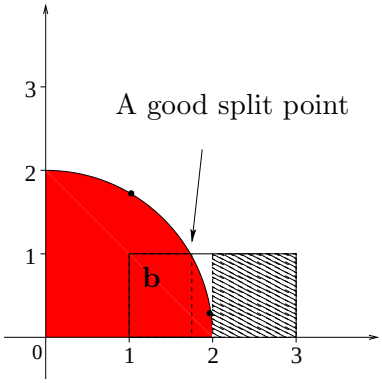


Figure 4.7: Example of a specific split operator based on the negation test.

represent the solutions). A good summary of these works, including a comparison of many of the algorithms proposed in [191, 203, 204, 205] can be found in [206].

As noted by [14], in order to apply the *negation test* the constraints need to be easily negated. This is the case, for example, of inequalities in universally quantified constraints but it is not the case in equations involving existentially quantified parameters.

For this reason, this type of problems has been studied from a different point of view. Bouchon-Meunier *et al.* [22] studied the limitation of modal mathematics for solving problems involving different quantified variables. As result they propose a classification of the problems involving quantifiers and modalities into *decidable/undecidable* and *polynomial/no polynomial* time complexity.

In [190], Shary shows that Modal Interval Analysis combined with Kaucher generalized interval arithmetic can successfully be applied to problems with existentially and universally quantified parameters, and Herrero [95, 96] presents some results of the application of modal interval analysis to control problems.

In the last years, the problem to solve linear and non-linear systems involving quantified parameters has interested many authors. Some examples are the works of Shary [190], Jaulin [107, 108], Popova [164, 165], and Goldsztejn [73, 74].

4.7 Solving Distance Constraints

As presented in the introduction, distance constraints play an important role in many areas like Robotics and Molecular biology. For this reason, many works in

these areas are related to this problem.

In [167, 197], for example, the authors solve systems of distances equations (without uncertainties/parameters) based on the theory of Cayley-Menger determinants [27, 139, 147]. The main idea is to use a matrix $\mathbf{A} = (\mathbf{a}_{ij})$ representing the distance between the points i and j. Unknown distance are represented by an interval $[0, \sigma]$ where σ represents the maximal possible distance[8]. The method iterates reducing and expanding the dimension of the problem in order to filter the ranges of the possible solution. A bisection phase is used when the algorithm does not reduce any range in the matrix. The main advantage of this approach is that it is independent of a reference frame.

Another approach introduced in [11] consists in using semantic domain decomposition (SDD) for distance constraints. It does not consider quantified parameters but it can be used when inequalities are involved in the problem. The main idea is to separate the initial domain of the variable into sub-domains for which the constraint (expressed as $f(x) = 0$) has a monotonic function $f(x)$. After that, and considering the monotonicity of $f(x)$, it is possible to compute the smallest box that verifies the constraint for each sub-domain. The set of solutions of the problem must to be inside the computed boxes. Preliminary results show that this approach increases the performance of the solving process in small problems. Experiments in large size problems have not be reported yet.

The basic algorithm presented in Section 4.6.1 is also a valid approach for solving distance constraints. As distance constraints have particular properties (as the single occurrence of the variables), filtering techniques (like 2B-consistency) can effectively reduce the domain of the variables to the smallest box that verifies a constraint. Anyway, it is important to remember the *locality* problem.

Although the 2B-consistency is a good strategy for filtering distance constraints, it remains a local consistency technique. Merlet [143] introduced a global filtering for systems of distance constrains inspired from the work of Yamamura *et. al.* [211]. The main idea is to perform a change of variables on the original system in order to generate a linear system of equations and inequalities. The simplex algorithm is then used to determine if the generated linear system admits a feasible region and to filter this original domain according to this feasible region (if exists).

[8] Actually, this upper bound can be easily computed because the distance between two points i, j cannot be greater than the sum of the partial well-known distances between intermediate points in the chain from i to j.

Another global filtering for distance equations, based on linearization has been proposed in [9, 12]. It introduces the algorithm *QuadDist* (based on the previous work of Lebbah *et. al.*[131]) which performs a linearization of the system in order to approximate parts of the space containing potential solutions. As same as Merlet[143], it uses the simplex algorithm to reduce the domain of the variables.

It is important to note that the effectiveness of the filtering techniques and solving process also depends on the selected reference frame. A good selection of the reference frame may drastically improve the solving process by reducing the complexity of the constraints and facilitating the domain reduction. Although this is a strategy frequently used in areas as Physics and Robotics[9], it is not well-known in other areas.

Example 4.7.1 *Consider the system of three distance equations graphically shown in Figure 4.8(a). It is described by the following equations:*

$$\begin{aligned} x_2^2 + y_2^2 &= 8^2 \\ x_3^2 + y_3^2 &= 10^2 \\ (x_2 - x_3)^2 + (y_2 - y_3)^2 &= 6^2 \end{aligned}$$

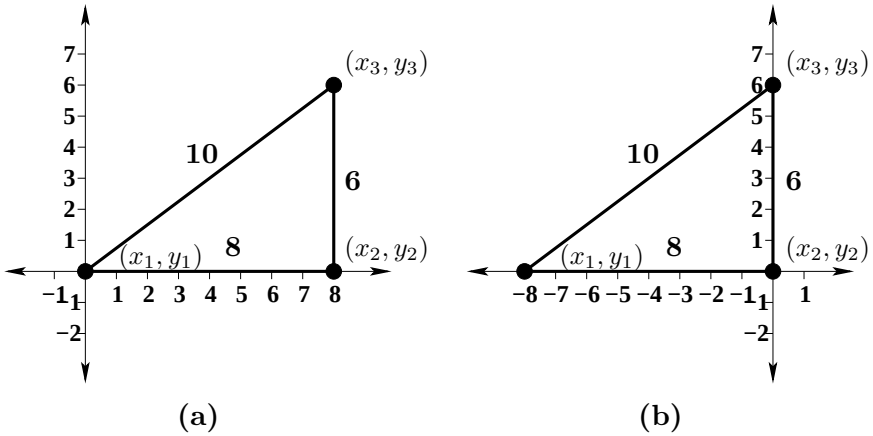


Figure 4.8: An example of two possible reference frames.

The same system, but using a reference frame centered in the point (x_2, y_2)*, is*

[9]In [75], for example, Gosselin *et. al.* showed that the polynomial of degree 12 (introduced by Merlet [140]) can be reduced to a polynomial of degree 6 by using an appropriate reference frame.

presented in Figure 4.8(b). It is described by the following equations:

$$\begin{aligned} x_1^2 + y_1^2 &= 8^2 \\ x_3^2 + y_3^2 &= 6^2 \\ (x_1 - x_3)^2 + (y_1 - y_3)^2 &= 10^2 \end{aligned}$$

A 2B-consistency technique applied to the first system reduces the domains of the variables to $(x_2, y_2) = ([-8, 8], [-8, 8])$ *and* $(x_3, y_3) = ([-10, 10], [-10, 10])$. *The same consistency technique applied to the second system reduces the domains of the variables to* $(x_1, y_1) = ([-8, 8], [-8, 8])$ *and* $(x_3, y_3) = ([-6, 6], [-6, 6])$. *In this example, a selected reference frame may reduce the domain of two variables by* 40%.

4.8 Conclusions

This chapter presented the combination of two important approaches for solving difficult problems over continuous domains: Constraint Programming and Interval Analysis.

Although Constraint Programming is a framework mainly used to solve combinatorial problems, it can also be used in presence of continuous domains. The development of techniques adapted to them have allow important improvements in the solving process of non-linear systems. The support given by Interval Arithmetic ensures the numerical reliability of computation while the set of tools provided by Interval Analysis (and Numerical Analysis in general) increases the performance of the solving process by providing proofs of existence and/or uniqueness of the solutions and new filtering techniques (as the interval Newton and Krawczyk operators).

A generic strategy for problem solving based on the Branch and Prune algorithm and a set of improvements to this basic strategy have been also discussed. The most important characteristic in this process is the *reliability*, which guarantees that the union of all the computed boxes contains all the solutions of the NCSP.

The problem of the uncertainty in the input data and its influence in the solutions of a system of equations has been also treated. This uncertainty has been introduced in the form of quantified parameters. As the problems involving inequalities or constraints with existentially quantified parameters frequently have

a solution set with a non-null volume, a representation of their solutions has been discussed. This representation introduced the concepts of inner approximation and outer approximation of the solution set of a NCSP, and they are the base of some works presented in the second part of this thesis. Thus, the next chapters focus on solving problems with uncertainties, and specifically on those involving distance constraints.

Part II

Contributions

Chapter 5

Separating Continua of Solutions

This chapter is based on the works presented in [79, 81]. The main objective is to describe the set of solutions of a distance equation system with uncertainties. As this set is composed by several continua of solutions, this chapter studies some tools to describe them.

The first approach consists of a three phases algorithm which performs an initial approximation of the set of solutions of the problem without considering the associated uncertainties. The main idea is to separate the initial space into a set of sub-spaces containing one isolated solution each, and (in a later step) study the influence of uncertainties in each of these sub-space.

As this approach has strong limits, the second part of this chapter studies a new methodology to approximate the disjoint continua of solutions of a problem. The main idea is to detect as many disjoint sub-spaces as possible based on a branch and prune algorithm with conditional bisection.

Section 5.1 presents a brief description of the first approach, while 5.2 presents some details of its implementation. Section 5.3 discusses the main drawback of this methodology and Section 5.4 presents a new approach which overcomes some of these drawbacks. Finally, Section 5.5 summaries the conclusions of the chapter.

5.1 A brief description

This section studies a first approach for solving a NCSP built from a set of distance constraints with uncertainties (as shown in Section 1.1).

For the types of problems presented in Section 1 describing physical situations, the uncertainties represent errors in the measurement of some parameters. It is natural to think that the NCSP without considering uncertain values has a finite set of isolated solutions, and that the uncertainties only disturb these solutions. Therefore, one of the methods for solving systems of distance equations (presented in Section 4.7) can be used to solve the NCSP resulting from replacing each parameter with interval value by the middle point of its associated interval, and to generate a set of isolated solutions. The advantage of such a process is to obtain more information about the geometry of the solutions, but no information about the continua of solutions is known.

Another possibility is to apply a branch and prune algorithm to generate a set of boxes that include all the continua of solutions, but this algorithm generates a lot of boxes without information about independent continua of solutions. Therefore, a subsequent task will be to apply a clustering algorithm (as in [203]) to recover the different solution sub-spaces.

The first approach discussed in this section (and based on the work of Touati [199]) combines different solving process into a three phases algorithm. The main idea is solving the NCSP without taking into account the uncertainties (by replacing each interval valued parameter by a real number). The aim is to identify a set of isolated solutions of the problem.

After that, a solution separation algorithm (SSA) is used in order to compute a set of independent sub-spaces containing only one isolated solution. The SSA computes the equation of the median plane between each pair of solutions.

Finally, for each solution found, we solve a new NCSP built from the original NCSP (with uncertainties) and a the set of inequalities given by SSA. A branch and prune algorithm (see section 4.6.1) is applied for solving each NCSP.

The best results of this methodology are obtained when the following two conditions are verified:

- The problem has a finite number of solutions $\{s_1, \ldots, s_n\}$, without taking into account the uncertainties.
- The problem has only independent continua of solutions around each initial solution, when the uncertainties are considered.

The first condition allows the separation of the initial domain into a set of sub-domains containing only one isolated solution, while the second one, guarantees the existence of only one solution sub-space in each sub-domain.

5.2 The basis algorithm

The first approach can be summarized in three steps:

1. Finding all solutions of the problem without considering the uncertainties.
2. Applying a division of the initial space into sub-spaces containing only one isolated solution each.
3. Considering the original problem (with uncertain values represented by interval parameters), applying a branch and prune algorithm on each generated sub-spaces in order to obtain a sharp approximation of the continuum of solutions included inside the sub-space.

Figure 5.1 graphically shows the three steps of the approach.

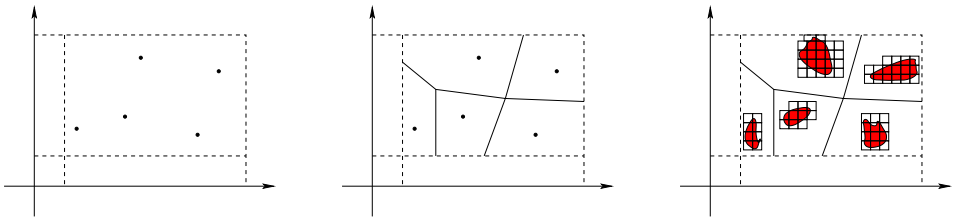


Figure 5.1: (a) A set of isolated solutions. (b) The result of the SSA applied for each point. (c) The result of a Branch and Prune applied to the problem with uncertainties on the sub-space of each solution.

Consider a NCSP P' obtained from the original NCSP P by replacing each parameter with interval value by the middle point of the interval. Let $S' = \{s_1, \ldots, s_n\}$ be the set of solutions of P', obtained from one of the methods for solving systems of distance equations described in Section 4.7.

The second step of the algorithm consists of applying a solution separation algorithm on S'. For each solution $s_i \in S'$, we calculate the equation of the median plane $Mp_{s_i,s_j}(x)$ between s_i and s_j, $s_i \neq s_j$, as shown in Algorithm 8.

The subspace generated by the conjunction of the inequalities obtained from the computed median planes for a given solution is similar to the sub-space of a Voronoi[1] diagram [7], as is shown in Figure 5.1(b).

[1]Actually, there is another way to to obtain the same sub-space by automatically computing the Voronoi diagram of the points (see CGAL in `http://www-sop.inria.fr/geometrica`).

Algorithm 8 : SSA(s, S'): *constraints*

$C' \leftarrow \emptyset$
for all solution s_i in $S' \backslash \{s\}$ **do**
  $Mp_{s,s_i}(x) \leftarrow (x - \frac{s+s_i}{2}) \cdot (\frac{s-s_i}{\|s-s_i\|}) \geq 0$
  $C' \leftarrow C' \cup Mp_{s,s_i}(x)$
return C'

In the third step, the original NCSP P is solved for each solution $s_i \in S'$, by considering the set of inequalities given by $SSA(s_i, S')$. A Branch and Prune algorithm is used to solve each new NCSP.

As noted by Benhamou and Goualard [14], using an inner box test[2] drastically improves the performance of the solving process, since each branching in the search tree exponentially increases the number of boxes (see Section 4.6.2).

The techniques discussed in Section 4.6.2 (based on the *negation test*) cannot be applied to this type of equation, because the continuum of solution is given by all the points x that verify a set of constraints of the form:

$$c(x) : (\exists a \in \mathbf{a})(\exists b \in \mathbf{b})(f(x, a) = b) \tag{5.1}$$

where $f(x, a)$ is the expression of the Euclidean distance between the points x and a, and these constraints are not easily reversed.

5.2.1 An inner box test

Considering the type of constraint presented in (5.1), it is possible to build an inner box test by evaluating the left side of the constraint using interval arithmetic and using following interval inclusion condition: $f(\mathbf{x}, \mathbf{a}) \subseteq \mathbf{b}$.

As the semantic of such a comparison implies that:

$$(\forall x \in \mathbf{x})(\forall a \in \mathbf{a})(\exists b \in \mathbf{b})(f(x, a) = b)$$

it can be used as an inner box test. The main drawback is that it is a very restrictive condition. For this reason we introduce another condition, which is less restrictive and allows one to find bigger inner boxes.

Proposition 5 *Let* $\mathbf{f}(\mathbf{x}, m(\mathbf{a}))$ *be the interval extension of* $f(x, a)$ *when* x *is replaced by its associated interval* $\mathbf{x}$ *and* a *is replaced by the middle point of* $\mathbf{a}$. *If*

[2] A test for detecting boxes which are totally included in a continuum of solutions.

$\mathbf{f}(\mathbf{x}, m(\mathbf{a})) \subseteq \mathbf{b}$, *then* $\mathbf{x}$ *is an inner box for the constraint.*

The proof is immediate using the definition of inner box and the semantics of the evaluation of $\mathbf{f}(\mathbf{x}, m(\mathbf{a}))$.

5.3 Drawbacks of the first approach

Preliminary experiments show that although it is possible to apply this approach for solving NCSP with uncertainties, its efficiency is very limited, mainly due to the following reasons:

- Sometimes, it is not possible to calculate all solutions to a NCSP P', either because the problem without uncertainties has no solutions (Figure 5.2(a)) or because P' has a infinite number of solutions (Figure 5.2(b)). In any of these cases, it is not possible to apply this approach.

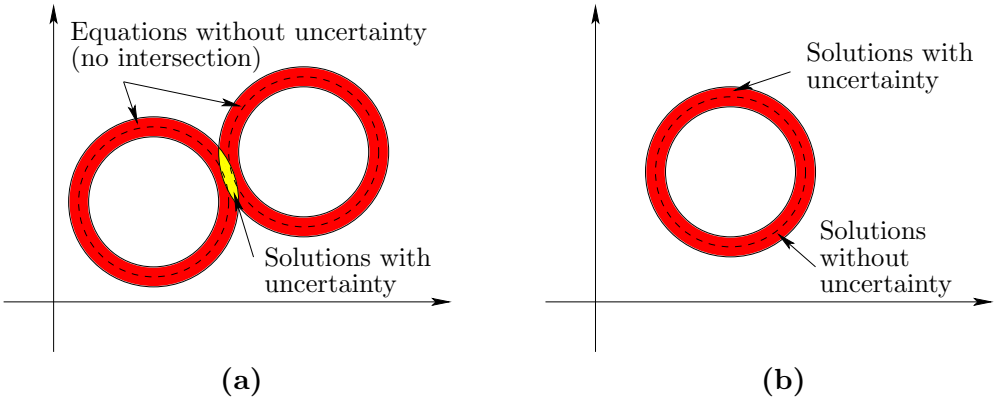


Figure 5.2: (a) A problem without solution when uncertainties are not considered (equations in dot line). (b) An under-constraint problem having an infinite number of solutions when the uncertainty is not considered (dot line).

- The approach considers that the number of continua of solutions in the NCSP with uncertainties P is equal to the number of isolated solutions of the NCSP without uncertainty P'. That is not necessarily true. Eventhough one or more isolated solutions can be found when solving P', that does not mean the problem with uncertainties has the same (or less) solution subspaces. In some cases, the number of continua of solutions can be greater[3] than the number of isolated solutions found (see Figure 5.3).

[3]It is not true if the initial solving phase considers both real and complex solutions, but this approach has not be studied here.

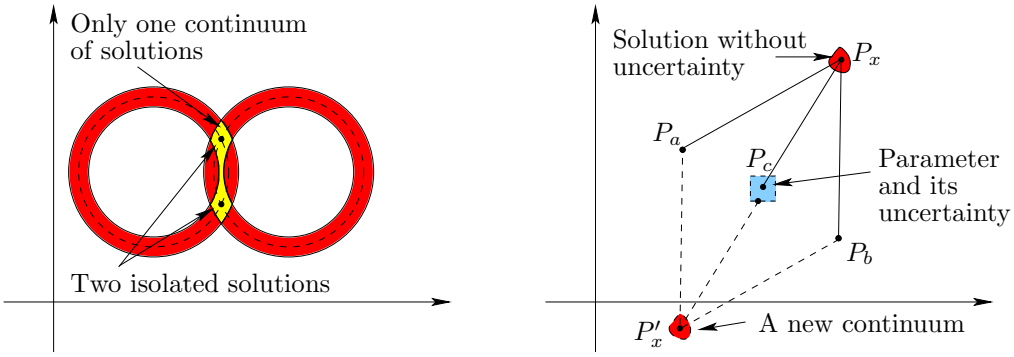


Figure 5.3: (a) Two isolated solutions become one continuum of solutions. (b) A single initial solution (P_x) obtained from a over-constrained problem (three distance constraints without uncertainty: $d(P_a, P_x)$, $d(P_b, P_x)$, $d(P_c, P_x)$), where P_a, P_b, P_c are fixed, becomes two continua of solutions (two zones, around P_x and P'_x) when the uncertainty of the point P_c is considered (small box around P_c).

- The inner boxes test presented in Section 5.2.1 is only a sufficient condition for an inner box, and its effectiveness is severely limited by the width of the interval representing fixed points. The bigger the interval, the worse the detection. Figure 5.4 shows the zone of detection for the constraint $c : (x-a)^2 + (y-b)^2 = c^2$. In the left side $a = 0$ and $b = 0$ (no uncertainty), and in the right side $a \in [-1, 1]$ and $b \in [-1, 1]$. In both $c \in [4, 5]$.

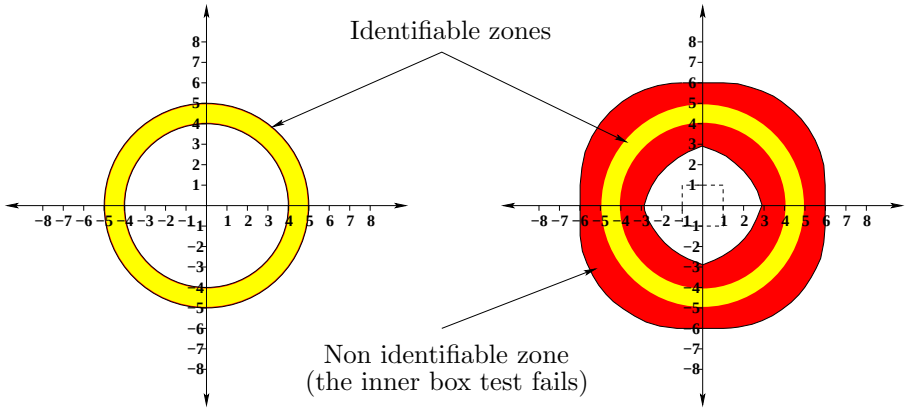


Figure 5.4: Examples of identifiable zones of a constraint with uncertainty.

5.4 Rough approximation of continua of solutions

An important drawback of the first approach (described in Section 5.1) is that it needs an initial set of isolated solutions to apply a space separation algorithm. This section introduces a new methodology for detecting different continua of solutions in NCSP involving distance constraints without using any additional information. This approach is based on a branch and prune algorithm with conditional bisection. The main idea is to forbid the separation of a continuum of solutions by analyzing the effect of a filtering technique after the application of a splitting operator. It is based on the following proposition:

Proposition 6 *Let $P = (X, D, C)$ be a NCSP, and D_a, D_b two sub-spaces such that $D = D_a \cup D_b$. Let $P_a = (X, D_a, C)$ and $P_b = (X, D_b, C)$ be two NCSPs derived from P by replacing the initial domain D by D_a and D_b, respectively. Let D'_a and D'_b be the resulting domains after applying a filtering technique[4] over P_a and P_b, respectively. If $D'_a \cap D'_b = \emptyset$ then no continuum of solutions is shared between D_a and D_b.*

The proof is based on the properties of the filtering technique. As shown in Chapters 2 and 4, constraint propagation eliminates parts of the domains of the variables that cannot belong to any solution of the problem. Therefore, if a continuum of solutions is shared between D_a and D_b then there exists at least a point x^* (solution) such that $x^* \in D_a$ and $x^* \in D_b$. If after applying a filtering technique we obtain $D'_a \cap D'_b = \emptyset$ then no solution $x^* \in D_a$ and $x^* \in D_b$ exists.

Algorithm 9 SOISS(P, D, ss)

$D' \leftarrow D$
while ss not empty **do**
  select and remove a *split_strategy* from ss
  $D_a, D_b \leftarrow apply\_split\_strategy(D)$
  $D'_a \leftarrow filtering(P, D_a)$
  $D'_b \leftarrow filtering(P, D_b)$
  if $D'_a \cap D'_b = \emptyset$ **then**
    return D'_a, D'_b
  else
    $D' \leftarrow Hull(D'_a, D'_b)$
return D'

[4]It can be any of the consistency techniques discussed in Section 4.3.

Algorithm 9 shows a generic implementation of the algorithm *Split Only Independent Solution Sets* (SOISS). It takes a problem P with initial domain D and a set of splitting strategies.

If one of the strategies produces two independent sub-spaces (that means no continuum of solutions is shared between them) then both sub-spaces are returned, else a hull approximation of the generated boxes is computed. This process is called *conditional bisection*.

Algorithm 10 presents a generic implementation of the Branch and Prune algorithm with conditional bisection, called Solver_SOISS. This algorithm has been implemented[5] in $C++$ using the IcosAlias library (a tool in development in the COPRIN project).

Algorithm 10 Solver_SOISS(P, D, ss)

```
Q ← filtering(P, D)
Sols ← ∅
while Q not empty do
  select and remove a box x' from Q
  if x' ≠ ∅ then
    Q_tmp ← SOISS(P, x', ss)
    if Q_tmp has only one box then
      Sols ← Sols ∪ Q_tmp
    else
      Q ← Q ∪ Q_tmp
return Sols
```

Note that SOISS is used not only for bisecting purposes, but also as filtering technique. As the sub-domains returned by SOISS have been filtered, it is not necessary to apply another filtering strategy in the main solver (as shown in the implementation of Solver_SOISS).

Table 5.1 presents some preliminary results on a set of benchmark problems[6]. A classic approach (based on a Branch and Prune algorithm[7] with precision 10^{-5}) is compared with Solver_SOISS.

A small uncertainty has been introduced in each equation (in the form of an interval valued parameter $\mathbf{p}$ with $w(\mathbf{p}) = 10^{-6}$). This uncertainty produces continua of solutions. For each problem, the number of solutions (without uncertainties) is

[5]Source code available in `http://www-sop.inria.fr/coprin/cgrandon/solver-soiss.cc`.

[6]The characteristics of these problems can be obtained in the website of the COPRIN project `http://www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/node1.html`.

[7]Implemented in IcosAlias v0.2 (see `http://www-sop.inria.fr/coprin/gchabert`).

Problem	**Classic Approach**			**Solver_SOISS**		
	Boxes	*Volume*	*Time (s)*	*Boxes*	*Volume*	*Time (s)*
Bronstein(4)	9	$2.32e^{-15}$	0.009	4	$7.91e^{-14}$	0.023
Cyclo(8)	276	$3.50e^{-14}$	0.248	8	$1.21e^{-6}$	0.568
Kearl11(16)	672	$1.06e^{-42}$	0.174	16	$1.74e^{-39}$	0.226
Kincox(2)	18	$9.73e^{-22}$	0.018	2	$2.05e^{-11}$	0.029
Nauheim(4)	233	$2.18e^{-40}$	0.681	4	$1.45e^{-20}$	17.409

Table 5.1: A Comparison between a classic approach and Solver_SOISS.

presented in parentheses.

Both processes are safe (no solution is lost), but the classic approach has no information about the geometry of the solutions (only a set of boxes is returned). Moreover, it depends on a given precision (in this case 10^{-5}), while Solver_SOISS stops when it cannot find new disjoint sub-spaces. Both algorithms have used the same filtering strategy (a 3B(w_1, w_2)-consistency with $w_1 = 10^{-3}$ and $w_2 = 10^{-4}$).

The *split_strategy* used in SOISS is a simple bisection in the middle (for each variable of the problem). No other optimization has been used.

The classic approach uses a *largest domain first* heuristic and a bisection in the middle as splitting strategy.

It is important to note that in all the problems, the number of boxes given by the Solver_SOISS algorithm was equal to the number of solutions of the original problem. A posterior analysis has shown that each box found contains one solution of the original problem (the problem without uncertainty). Although this is the best behavior of the algorithm, it may be difficult to obtain because it depends on the given splitting strategies (in a more general problem, a simple bisection in the middle is not always the best strategy, because the continua of solutions have not necessary a symmetry w.r.t. this point).

Anyway, the identification of independent zones of the space containing potential solutions of the problem may be of interest for many problems (as for example, in robotics). Chapter 6 presents an application in robotics of this algorithm.

5.5 Conclusions

This chapter presented a methodology for solving NCSP with uncertainties based on a three phases algorithm. This algorithm addresses to problems in which isolated solutions can be computed when the uncertainties are not considered.

In these cases, several sets of boxes (around each initial isolated solution) are returned by a branch and prune algorithm. In order to reduce the number of boxes generated by this algorithm a new inner box test (for detecting boxes containing only solutions of the problem during the solving process) has been also introduced.

We notice that the performance of this test is very limited when the parameters with interval values participate in the interval evaluation of the function (left-side of the equation in the constraint). Other strategies for improving the detection of inner boxes are studied in Chapter 7.

It is important to note that the returned clusters (sets of boxes) have no additional information about the geometry of the solutions. In other words, in order to guarantee that a median plane produces an appropriated separation of two continua of solutions an additional analysis must be performed[8].

The drawbacks of this approach (presented in Section 5.3) motivate the introduction of a new strategy for solving these types of problem. The advantage of this strategy is that it can be used in different situations (not only for NCSP involving distance constraints) for detecting disjoint continua of solutions without using a later clustering strategy.

Some preliminary results show that it can effectively improve the information about the geometry of the solutions. Moreover, the use of filtering techniques to separate continua of solutions guarantees the appropriate separation of the initial domain.

As future work, it can be interesting to combine the first approach with the Solver_SOISS algorithm into a four phases algorithm. The first phase identifying a set n of isolated solutions, while in a second phase, the Solver_SOISS algorithm may identify a set m of disjoint sub-spaces containing potential solutions. In this case, the use of a space separation algorithm (using a median plane) can be restricted only to the sub-spaces $k \in m$ containing more than one isolated solution.

[8]For example, by searching for solutions in the equation of the median plane computed for two isolated solutions (when uncertainties are considered), or proving that any intersection between a box included in one cluster and a box included in another cluster produces an empty set.

Chapter 6

An application in Robotics

This chapter is based on the works presented in [77, 78]. It focuses on a combination of symbolic and numerical methods for handling uncertainties in a particular class of parallel robots.

The main idea is to compute, for given uncertainties on parameters and measurements, a certified enclosure of the set of all possible poses of a robot. Pose determination is done by solving the *direct Kinematics*, which relies on the geometric parameters and measurements.

A model involving uncertainties (represented by intervals) is considered and several methods combining interval analysis and constraint programming are compared.

Some experiments have been performed with the CATRASYS (Cassino Tracking System) measuring system, which is a wire tracking system used for determining of poses of objects in the space. It is well modeled as a 3-2-1 parallel manipulator.

Section 6.1 presents an introduction to the pose determination problem and some measuring system commonly used to solve it. In Section 6.2, a formal model of the 3-2-1 parallel robot is presented. Section 6.3 introduces the different approaches for solving the problem that are combined in the final algorithm. Section 6.4 describes the CATRASYS measuring system and presents some preliminary results. Finally, Section 6.5 presents the conclusions of the chapter.

6.1 Introduction

An object's pose usually represents the rigid body transformation between a moving frame attached to this object and a base fixed frame. The problem of pose identification of a rigid body in the space has been studied from different points of view, from theoretical approaches [3] to numerical algorithms [202].

Several measuring systems are widely used to determine the pose of a rigid body. Current technologies for pose determination include vision systems, photogrammetry, theodolite, laser interferometry, magnetic tracking systems, stereo optical image registration, and acoustic methods.

The above-mentioned systems can be classified according to the measuring principle and used technology, but many of them are rather complex and expensive. The measuring principle of most of the above-mentioned systems is based on trilateration or triangulation techniques. Trilateration and triangulation determine the relative position between points by using the geometry of triangles or tetrahedra. Triangulation uses measurements of both distances and angles, whereas trilateration uses only distance measurements.

Measuring systems can also be classified according to their features, such as accuracy, resolution, cost, measurement range, portability, and calibration requirements. Laser tracking systems exhibit good accuracy, which can be less than 1mm if the system is well calibrated. Unfortunately, they are rather expensive, their calibration procedure is time consuming, and they are sensitive to the environment. Vision systems have an accuracy of 0.1mm, they are low cost portable devices but their calibration procedure can be complicated. Wire-based systems may have an accuracy of 0.1mm, they are also low cost portable devices but capable of measuring large displacements. Moreover, they exhibit a good compromise between measurement range, cost and operability.

Wire-based tracking devices consist of a fixed base and a platform connected by six wires whose tension is maintained, while the platform is moved by pulleys and spiral springs on the base, where a set of encoders give the wires' length.

Figure 6.1 presents a typical wire-based tracking device seen as a *3-2-1 parallel robot* (a simplified version of a general 6-Degrees of freedom parallel manipulator, with a set of 3 wires attached to the same point H of the platform, and a another set of 2 wires attached to a point F, and a single wire attached to a point Q).

Computing an object's pose using this robot means solving the *direct Kinematics* of the robot, based on geometric and measured information. Geometric

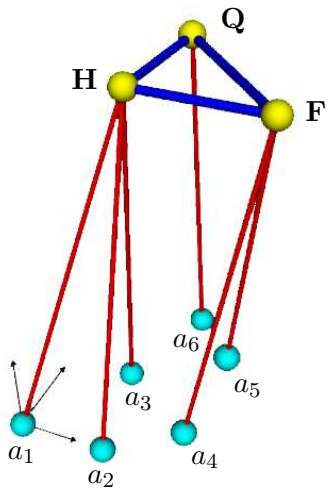


Figure 6.1: A typical wire-based tracking based on a 3-2-1 parallel robot.

information is in general easily obtained from the design of the robot while measured information must be obtained from built-in sensors.

Uncertainties, which appear in these parameters are namely due to measurements and manufacturing, and have to be taken into account when certified results are required.

Algebraic methods for solving the Direct Kinematics for the 3-2-1 parallel robot using the formulation for trilateration as been proposed in [68, 99, 158]. The answer of these methods is a good approximation of the solution, but they do not consider uncertainties in the measurement.

Numerical method based on interval analysis [52, 143] have shown to give certified results when applied to system of equations with uncertainties, but due to a wrapping effect, and to the interval natural extension, using only interval arithmetic for evaluating the expressions of the direct kinematic model does not produce good results. For this reason, it is natural to combine interval analysis tools with the set of tools provided by constraint programming (specifically, with the consistency techniques discussed in Chapter 4.3).

The rest of this chapter shows how the combination of these strategies greatly enhances the quality of the results in the computation of the direct Kinematics of a 3-2-1 parallel robot, and how symbolic computation improves the solving process of numerical methods.

6.2 A Formal Model

As shown in Figure 6.1, the 3-2-1 parallel robot considered here, consists of two solids (the basis and the mobile platform) linked through six linear rods of variable lengths.

The direct kinematic model expresses how the lengths of the legs are related to the position and the orientation of the mobile platform, through six distance equations (commonly presented in a vectorial form):

$$\|\mathbf{a}_1 - H\|^2 = \mathbf{d}_1^2 \qquad \|\mathbf{a}_2 - H\|^2 = \mathbf{d}_2^2 \qquad \|\mathbf{a}_3 - H\|^2 = \mathbf{d}_3^2 \tag{6.1}$$

$$\|\mathbf{a}_4 - F\|^2 = \mathbf{d}_4^2 \qquad \|\mathbf{a}_5 - F\|^2 = \mathbf{d}_5^2 \tag{6.2}$$

$$\|\mathbf{a}_6 - Q\|^2 = \mathbf{d}_6^2 \tag{6.3}$$

When studying a 3-2-1 robot, to take into account its specificity, it is classical to represent the position and the orientation of the mobile platform through the nine unknown coordinates of the three attachment points H, F, and Q. As the geometry of the mobile platform is known, the direct kinematic model is completed by the following three distance equations:

$$\|H - F\|^2 = \mathbf{d}_{HF}^2 \tag{6.4}$$

$$\|H - Q\|^2 = \mathbf{d}_{HQ}^2 \tag{6.5}$$

$$\|F - Q\|^2 = \mathbf{d}_{FQ}^2 \tag{6.6}$$

Notice that the parameters of the system (fixed points $a_1, \ldots, a_6$, and distances) have been considered as interval values, which represent the uncertainties in the measurements and manufacturing process.

6.3 The Solving Process

This section describes different parts of the solving process that will be combined in a symbolic/numerical algorithm for solving the system of equations presented in Section 6.2.

6.3.1 Algebraic Reduction

The method described here is based on the works by Ceccarelli [28] and Ottaviano [162]. Considering the formulation given in Section 6.2, the Forward Kinematics

of the 3-2-1 parallel robot can be algebraically reduced using three consecutive trilateration operations.

Indeed, according to Figure 6.2(a), giving the wire lengths d_1, d_2, and d_3, there are two possible mirror locations for point H with respect to the plane defined by points a_1, a_2, and a_3. Once one of these two solutions is chosen, a_4, a_5, and H define the second tetrahedron with known edge lengths. Again, there are two possible mirror locations for F, in this case with respect to the plane defined by a_4, a_5 and H, as shown in Figure 6.2(b). Finally, after choosing one of the two solutions, a_6, H, and F define another tetrahedron with known edge lengths. In this case there are two possible mirror locations for Q with respect to the plane defined by a_6, H, and F (Figure 6.2(c)).

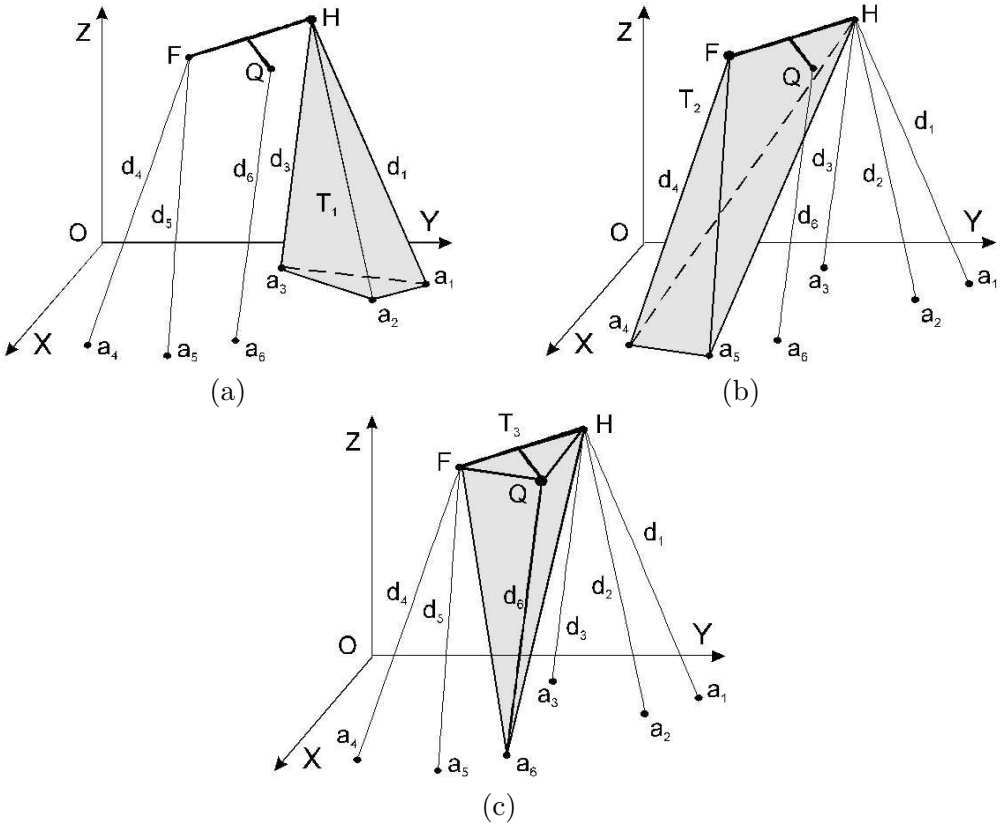


Figure 6.2: Three consecutive trilateration operations to compute the position of H, F, Q: (a) Tetrahedron T_1, (b) Tetrahedron T_2, and (c) Tetrahedron T_3.

According to Section 4.7 (*the importance of choosing an appropriate reference frame*) and without lost of generality we can assume our reference frame to be

based on a_1, a_2, and a_3, i.e. that

$$x_{a_1} = y_{a_1} = z_{a_1} = y_{a_2} = z_{a_2} = z_{a_3} = 0$$

So, algebraic transcription of the above described method computes a closed-form of the solution in three steps:

1. equations in (6.1) become

$$\begin{array}{rcrcrcl} x_H^2 & + & y_H^2 & + & z_H^2 & = & d_1^2 \\ (x_H - x_{\mathbf{a}_2})^2 & + & y_H^2 & + & z_H^2 & = & d_2^2 \\ (x_H - x_{\mathbf{a}_3})^2 & + & (y_H - y_{\mathbf{a}_3})^2 & + & z_H^2 & = & d_3^2 \end{array} \tag{6.7}$$

   and allow us to easily express x_H, y_H, and two solutions for z_H,

2. Given x_H, y_H, z_H, equations in (6.2) and (6.4), the coordinates x_F, y_F, and z_F are then expressed by the equations

$$\begin{array}{rcrcrcl} (x_F - x_{\mathbf{a}_4})^2 & + & (y_F - y_{\mathbf{a}_4})^2 & + & (z_F - z_{\mathbf{a}_4})^2 & = & d_4^2 \\ (x_F - x_{\mathbf{a}_5})^2 & + & (y_F - y_{\mathbf{a}_5})^2 & + & (z_F - z_{\mathbf{a}_5})^2 & = & d_5^2 \\ (x_F - x_H)^2 & + & (y_F - y_H)^2 & + & (z_F - z_H)^2 & = & d_{HF}^2 \end{array} \tag{6.8}$$

3. and similarly, x_Q, y_Q, and z_Q are expressed by

$$\begin{array}{rcrcrcl} (x_Q - x_{\mathbf{a}_6})^2 & + & (y_Q - y_{\mathbf{a}_6})^2 & + & (z_Q - z_{\mathbf{a}_6})^2 & = & d_6^2 \\ (x_Q - x_H)^2 & + & (y_Q - y_H)^2 & + & (z_Q - z_H)^2 & = & d_{HQ}^2 \\ (x_Q - x_F)^2 & + & (y_Q - y_F)^2 & + & (z_Q - z_F)^2 & = & d_{FQ}^2 \end{array} \tag{6.9}$$

Theoretically, successive substitutions of the coordinates of H and of F return eight different solutions for the nine expressions of the coordinates of the point Q in terms of the parameters (coordinates of the a_i and distances). Obviously, the final expressions for the coordinates have multiple occurrences of the parameters $(x_{\mathbf{a}_i}, y_{\mathbf{a}_i}, z_{\mathbf{a}_i}, d_i^2)$.

6.3.2 Interval Evaluation

Numerical solutions of the direct kinematic model are computed by evaluating at each of the above steps the symbolic expressions after substitution of the numerical values of the parameters, and of the results of the previous steps.

Due to measurement uncertainties and to the inaccuracy of manufacturing and assembly, we only have approximate values and bounded errors for the parameters. In this context, we use an interval representation, and we are interested in getting certified enclosing 3D-boxes approximating the positions of the points H, F and Q for each of the solutions. These points describe the position and orientation of the platform containing the robot end-effector.

The classical way to compute the corresponding intervals is to perform the evaluation of the symbolic expressions using the interval values of the parameters and interval arithmetic. For avoiding numerical errors when evaluating huge expressions, this numerical evaluation is done at each of the three steps described above.

Actually, each sub-system of three distance equations is solved in the same way (trilateration), vanishing square terms by subtracting the initial equations and generating linear expressions. A polynomial of degree 2 is then solved to obtain the solutions of the initial system. The process is performed as follows:

- Considering the generic system presented in (6.8).
- Subtracting the second equation from the first one to obtain a linear equation $L_1(x_F, y_F, z_F)$, as follows:

$$2x_F(x_{\mathbf{a}_5} - x_{\mathbf{a}_5}) + 2y_F(y_{\mathbf{a}_5} - y_{\mathbf{a}_5}) + 2z_F(z_{\mathbf{a}_5} - z_{\mathbf{a}_5}) + K = 0$$

  with $K = x_{\mathbf{a}_4}^2 + y_{\mathbf{a}_4}^2 + z_{\mathbf{a}_4}^2 - d_4^2 - (x_{\mathbf{a}_5}^2 + y_{\mathbf{a}_5}^2 + z_{\mathbf{a}_5}^2 - d_5^2)$.
- Subtracting the third equation from the first one to obtain another linear equation $L_2(x_F, y_F, z_F)$, similar to the last one.
- Getting $x_F(z_F)$ and $y_F(z_F)$ from L_1 and L_2.
- Replacing x_F by $x_F(z_F)$ and y_F by $y_F(z_F)$ in the third equation and solving a quadratic polynomial $p(z_F) = az_F^2 + bz_F + c$.

The classic expression $z_F = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ is then used to obtain the solutions of $p(z_F)$. The expressions are manipulated to minimize the number of occurrences of the variables (in order to improve their interval evaluation). As the same process can be applied to obtain a polynomial $p'(x_F) = a'x_F^2 + b'x_F + c'$ (starting from $y_F(x_F)$ and $z_F(x_F)$) or $p''(y_F) = a''y_F^2 + b''y_F + c''$ (starting from $x_F(y_F)$ and $z_F(y_F)$) without additional complexity, we compute the solutions of the system using the three possible polynomials and intersect the obtained solutions.

Unfortunately, results are strongly overestimated, in term of width of the obtained intervals as illustrated by the numerical example of Table 6.2. This is a well known drawback of interval methods of accumulating and propagating uncertainties and of not properly considering multiple occurrences of the same variables when evaluating an expression.

Summarizing, even though a explicit formula to compute the coordinates of a given point is available, it is not a good approach in general. The computed expression will be optimal only if each of its terms has a single occurrence.

6.3.3 Constraint Programming

To overcome the overestimation of the interval evaluation, classical methods of Constraint Programming have been implemented and tested. The SOISS algorithm (introduced in Section 5.4) has been used in order to avoid the separation of a continuum of solutions.

Experiments have been carried out using an algorithm with four different levels:

1. The basic level, tested for comparison purpose, consists of an interval evaluation of symbolic expressions, at each of the three steps. It only uses the Algebraic Reduction combined with Interval Evaluation.

2. Second level applies the Solver_SOISS algorithm using a 2B-consistency as filtering technique to the solutions after each step and a 2B-consistency to the whole system after the third step.

3. Third level applies the same strategy as the previous one but using 3B-consistency instead of 2B-consistency.

4. The last level is an enhancement of the previous, computing a sharp approximation of the solution sets using a branch and prune algorithm for each sub-space given by the Solver_SOISS algorithm. A rough precision has been set in order to reduce the number of returned boxes. The inner box test for distance constraints presented in Section 5.2.1 (that is, the condition $\mathbf{f}(\mathbf{x}, m(\mathbf{a})) \subseteq \mathbf{b}$, with $\mathbf{b} = \mathbf{r}^2$) has been also included.

6.4 The CATRASYS measuring system

CaTraSys (Cassino Tracking System) is a measuring system, which has been conceived and designed at LARM (LAboratory of Robotics and Mechatronics) in

Cassino, to determine the pose of a rigid body by using trilateration. Details of CaTraSys and its operation are reported in [28] and [198]. CaTraSys system is composed of a mechanical part, an electronics/informatics interface unit, and a software package. The mechanical part consists of a fixed base, which has been named as Trilateral Sensing Platform, and a moving platform, which has been named as end-effector for CaTraSys (see Figures 6.3 and 6.4).

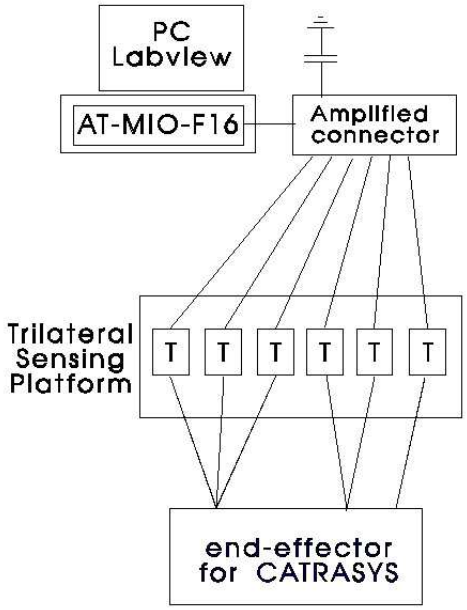


Figure 6.3: CaTraSys: A scheme for the design and operation.

It is a coupling device: it connects the wires of the six transducers to the extremity of a moving system. It allows the wires to track the system while it moves. Signals from wire transducers are fed though an amplified connector to the electronic interface unit, which consists of a Personal Computer for data analysis.

The wire transducers in the built prototype are of potentiometric type. They have a working range of 2500mm and continuous resolution (see [29]). A torsional spring, a pulley for the wire and a potentiometer are fixed on a common shaft. The output transducer signal is proportional to the length of the wire and it is expressed in Volt. Tension of the wire is ensured through the torsional spring. The tension of the wire is 0.08 N. The position of each attachment point on the Trilateral Sensing Platform has an uncertainty of ± 0.1mm.

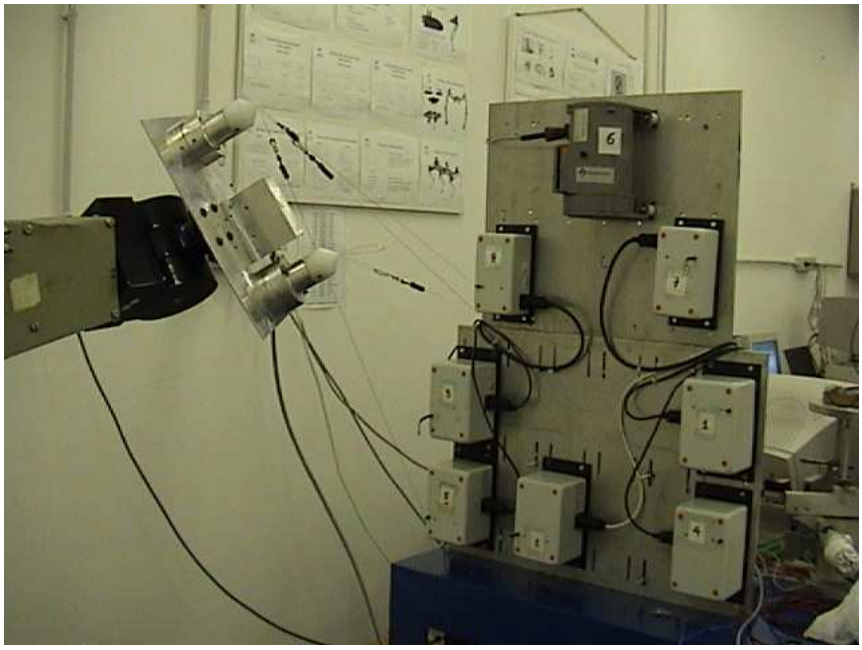


Figure 6.4: CaTraSys: A built prototype with PUMA robot at LARM in Cassino. Six of the white boxes (on the right) are linked to the platform by wires. The boxes have sensors to measure the length of the wires. In the image, the platform is placed on the terminal effector of the PUMA robot (a mechanical arm).

6.4.1 Experimental Results

A set of measurements of the distances and the geometric parameters were done in order to compute the direct Kinematics of CaTraSys for a given position. Table 6.1 shows the set of interval values for the coordinates (x, y, z) of the points $a_1, \ldots, a_6$, and the set of distances corresponding to the model presented in section 6.2.

Table 6.2 below shows the results (1 selected solution) obtained for the numerical example described in Table 6.1.

The precision vector is the vector of the half-widths of the intervals computed for the respective coordinates of H, F and Q. In other words, if $([0, 2], [2, 6], [0, 1])$ is the final box computed for the point ξ, then its corresponding precision vector is $(1, 2, 0.5)$ (the distance between the middle point and the upper bound for each coordinate).

Levels 1, 2, and 3 compute a solution in the form $s \pm \epsilon$ where s represents the coordinates of the points H, F and Q, and ϵ is the precision vector. Partial volumes are the respective volumes of the corresponding boxes. Levels 4 and 4^* compute a representation of the solution space through a set of boxes with a given precision

$$
\begin{array}{lcllcllcl}
x_{a_1} & = & 0 & y_{a_1} & = & 0 & z_{a_1} & = & 0 \\
x_{a_2} & = & [1000, 1001] & y_{a_2} & = & 0 & z_{a_2} & = & 0 \\
x_{a_3} & = & [799, 800] & y_{a_3} & = & [1199, 1200] & z_{a_3} & = & 0 \\
x_{a_4} & = & [1800, 1801] & y_{a_4} & = & [400, 401] & z_{a_4} & = & [199, 200] \\
x_{a_5} & = & [2099, 2100] & y_{a_5} & = & [900, 901] & z_{a_5} & = & [99, 100] \\
x_{a_6} & = & [1300, 1301] & y_{a_6} & = & [2199, 2200] & z_{a_6} & = & [200, 201]
\end{array}
$$

$$
\begin{array}{lcl}
d_1 & = & [1100, 1110] \\
d_2 & = & [900, 910] \\
d_3 & = & [1203, 1213] \\
d_4 & = & [855, 865] \\
d_5 & = & [801, 811] \\
d_6 & = & [872, 882]
\end{array}
\qquad
\begin{array}{lcl}
d_{HF} & = & [1489, 1490] \\
d_{HQ} & = & [1256, 1257] \\
d_{FQ} & = & [1799, 1800]
\end{array}
$$

Table 6.1: Measurements for the parameters of the robot's model (lengths in mm), provided by the LARM in Cassino, Italy.

Level	**Precision vector**	**Time**
1	(10.4, 17.2, 22.5);(125, 86, 155);(1350, 1350, 1400)	0.001
2	(10.4, 17.2, 16.2);(45.6, 73.2, 15.5);(637, 395, 539)	0.002
3	(10.3, 9.57, 6.74);(23.5, 32.8, 7.22);(56.6, 24.3, 25.4)	0.127
4	(10.3, 9.57, 6.74);(23.5, 32.8, 7.22);(56.6, 23.1, 25.4)	4.041
4*	(10.3, 9.57, 6.74);(23.5, 32.8, 7.22);(56.5, 21.5, 25.4)	142.9

Level	**Partial volumes**	**Total volume**
1	$(3.2244 \times 10^4, 1.3283 \times 10^7, 2.0331 \times 10^{10})$	$8.7e^{21}$
2	$(2.3222 \times 10^4, 4.15187 \times 10^5, 1.0835 \times 10^9)$	$1.1e^{19}$
3	$(5.321 \times 10^3, 4.4438 \times 10^4, 2.78634 \times 10^5)$	$6.6e^{13}$
4	$-($ 4,438 *boxes* $)-$	$1.1e^{11}$
4*	$-($ 158,591 *boxes* $)-$	$6.9e^9$

Table 6.2: Numerical results of the different levels of our algorithm. The first level uses only interval evaluation, while the remaining levels combine different consistency techniques. Levels 4 and 4* work in $\mathbb{R}^9$.

of 10mm and 5mm, respectively. For these levels, the computed precision vector is obtained by computing a rough (box) approximation (a box which contains all the boxes returned by the solver). The partial volume of these levels only shows the number of boxed returned by the solver. Figure 6.5 shows the computed results for the points H, F and Q with the first three levels. Figure 6.6 is a zoom on the points **H** and **F**, showing also the set of boxes computed by the last level.

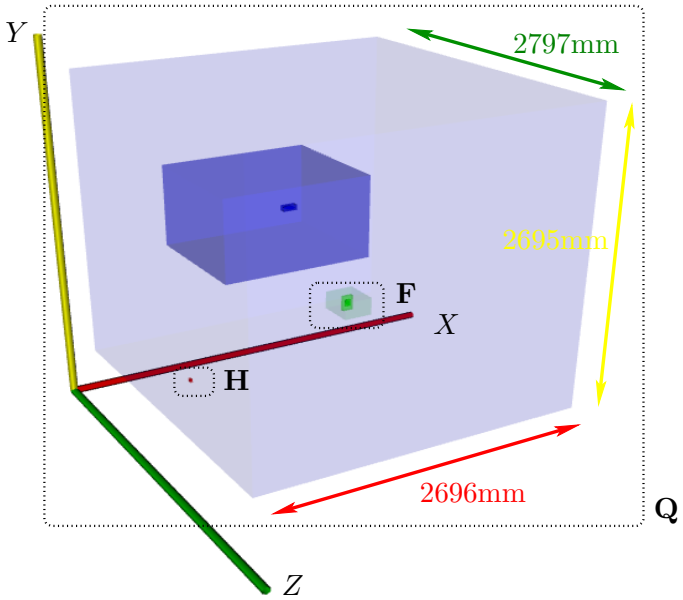


Figure 6.5: Computed results for the points **H** (red boxes), **F** (green boxes), and **Q** (blue boxes) using the first three levels of the algorithm. Notice the difference between the evaluation of the explicit formula (biggest box) and the results using filtering techniques.

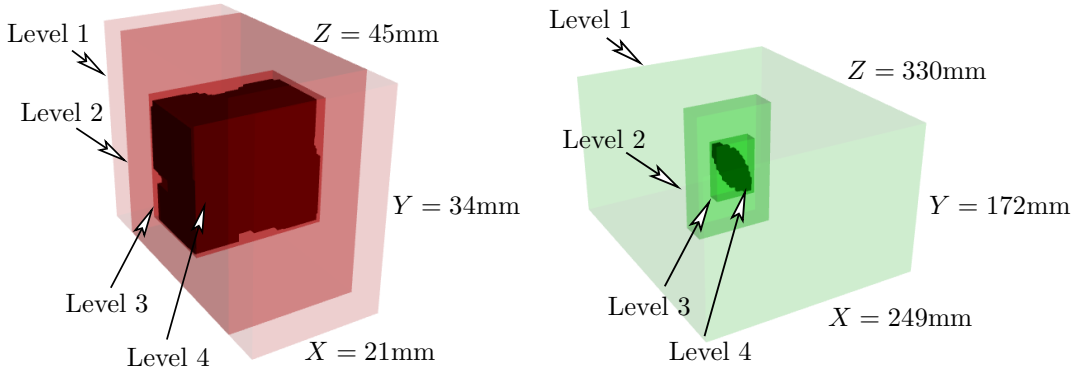


Figure 6.6: Zoom on points **H** and **F**. The darkest zone (given by the Level 4) is a sharp approximation containing several boxes.

6.5 Conclusions

This chapter presented a wire-parallel measuring device and an efficient algorithm for handling uncertainties in the computation of its Direct Kinematics. Certified results of a pose determination using this measuring device and computed by different levels of this algorithm have also been shown.

The algorithm, which combines a phase of formal resolution, and a phase of interval evaluation and propagation is able to handle uncertainties in the direct Kinematics problem of a whole class of parallel robot.

The experimental studies emphasizes the problem of interval evaluation of a model and shows that specific filtering algorithms are interesting to provide realistic sharp results.

Some important advantages of the presented method are:

- *a better description of the geometry of the solutions*, mainly given by the combination of the formal approach (algebraic reduction) and the SOISS algorithm.
- *certified results*, guaranteed by the use of interval arithmetic for the evaluations and the intermediate operations.
- *an improved approximation*, mainly given by the use of filtering techniques and constraint propagation.

It is important to note that a four level algorithm not only provides a way to compare different techniques, but also a choice between *required precision* and *time limitation*. The first level is the fastest (but it is not so sharp), while the last level is the sharpest (but time consuming).

Chapter 7

Improving the inner box detection

This chapter is based on the works presented in [79, 80, 82]. As shown in Chapter 6, a combination of symbolic and numerical approaches can effectively improve the solving process by providing more information about the geometry of the solutions, and better outer-approximation of the different continua of solutions. It overcomes some of the limitations of the basis algorithm presented in Chapter 5, but it remains another one: *the detection of inner boxes*.

A good detection of inner boxes prevents the division of a box (containing only solutions of the problem) into a lot of boxes, avoiding the waste of time due to this process. In this chapter, different approaches for detecting inner boxes in NCSP involving distance constraints with uncertainties are studied and compared.

The outline of the chapter is as follows: Section 7.1 presents an introduction and the motivation of such a test. In Section 7.2 the problem of quantifier elimination is introduced. Section 7.3 shows how existentially quantified parameters can be eliminated from the evaluation of a distance constraint in order to use classic interval arithmetic for detecting inner boxes. In Section 7.4, another approach based on generalized intervals for building an inner box test is discussed. The explanation of how these tests can be used for detecting inner boxes in *systems of distances equations with uncertainties* is given in Section 7.5. A comparison between the approach based on a specific quantifier elimination algorithm and that based on generalized interval evaluation is presented in Section 7.5.1. Based on these approaches, Section 7.6 presents an optimal test for distance constraints. Finally, Section 7.7 presents the conclusion of the chapter.

7.1 Introduction

Sometimes, an outer (box) approximation of different continua of solutions is not enough for solving a given problem. A good example is the workspace determination of a parallel robot. In this problem, given a set of control parameters and a robot design, one is interested in the set of effective positions that the robot end-effector can reach. For this reason, a tight description of the workspace (given by a branch and prune algorithm with a high precision) is for far a better approach.

It is important to note that the performance of such an algorithm is strongly influenced by the inner box detection process. Without an inner box test, such a branch and prune algorithm will bisect again and again the boxes included inside the continua of solutions, leading to inefficient computations and providing enclosures that are either prohibitively verbose or poorly informative.

Interval arithmetics can be successfully used for building such a test in some types of quantified constraints. For example, an interval evaluation of a function $f(x, y) : 3x^2 - 2y + 9$ with $x \in [-1, 3]$ and $y \in [-2, 4]$ can be used for detecting an inner box in a constraint $f(x, y) > 0$, because

$$\mathbf{f}([-1, 3], [-2, 4]) = 3 \cdot [-1, 3]^2 - 2 \cdot [-2, 4] + 9 = [1, 40]$$

As the evaluation of the function f is inside the interval $[1, 40]$, that means

$$(\forall x \in [-1, 3])(\forall y \in [-2, 4])(3x^2 - 2y + 9 > 0)$$

In some quantified distance constraints, this process can also be successfully used, when existentially quantified parameters are not taken into account in the evaluation of the function. For example, consider the following constraint:

$$c_{\mathbf{r}}(x) : (\exists r \in \mathbf{r})(x_1^2 + x_2^2 = r^2)$$

with $\mathbf{x}_1 = [1, 2]$, $\mathbf{x}_2 = [-1, 1]$, and $\mathbf{r} = [1, 3]$. Let $f(x_1, x_2) = x_1^2 + x_2^2$ be a function and consider the interval evaluation of $\mathbf{f}(\mathbf{x}_1, \mathbf{x}_2) : ([1, 2]^2 + [-1, 1]^2) = [1, 5]$. From the point of view of interval arithmetics, we have the following semantic interpretation:

$$(\forall x_1 \in \mathbf{x}_1)(\forall x_2 \in \mathbf{x}_2)(\exists z \in [1, 5])(x_1^2 + x_2^2 = z)$$

In particular, if $[1,5] \subseteq \mathbf{r}^2$ the following proposition is also verified:

$$(\forall x_1 \in \mathbf{x}_1)(\forall x_2 \in \mathbf{x}_2)(\exists r \in \mathbf{r})(x_1^2 + x_2^2 = r^2)$$

So, the inclusion $\mathbf{x}_1^2 + \mathbf{x}_2^2 \subseteq \mathbf{r}^2$ is a sufficient (and necessary, in this case) condition for the detection of an inner box.

In a more general case (a general quantified distance constraint with uncertainties), the interval evaluation is less effective, because existential quantified parameters are inside the evaluated function. For example, consider the following two dimensional distance constraint:

$$c_{\mathbf{a},\mathbf{r}}(x) : (\exists a \in \mathbf{a})(\exists r \in \mathbf{r})((x_1 - a_1)^2 + (x_2 - a_2)^2 = r^2) \tag{7.1}$$

with $\mathbf{x}_1 = [3,6]$, $\mathbf{x}_2 = [-1,1]$, $\mathbf{a}_1 = [-1,1]$, $\mathbf{a}_2 = [-1,1]$, and $\mathbf{r} = [4,5]$. If we consider the function $f(a,x) = (x_1 - a_1)^2 + (x_2 - a_2)^2$, the interval evaluation of $\mathbf{f}(\mathbf{a},\mathbf{x})$ is

$$([3,6] - [-1,1])^2 + ([-1,1] - [-1,1])^2 = [4,53]$$

and the inclusion $[4,53] \subseteq \mathbf{r}^2$ is not true, so the test fails.

A classic interval evaluation cannot prove that the interval vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$ is inside of the solution set of the constraint, but it is. This fault is not due to interval arithmetics but to a conceptual problem in its application: *existentially quantified parameter have been considered as universally quantified ones.* In other words, the evaluation $\mathbf{z} = \mathbf{f}(\mathbf{a},\mathbf{x})$ verifies $(\forall x \in \mathbf{x})(\forall a \in \mathbf{a})(f(a,x) \in \mathbf{z})$ which is not the semantics needed.

This chapter studies several strategies to detect inner boxes in quantified distance constraints when existentially quantified parameters are involved in the expression of the function.

7.2 Quantifier Elimination

Informally, the basic motivation of the quantifier elimination is to *eliminate* unwanted variables from an algebraic description of some situation. These variables may represent parameters of a model, or real quantities that cannot be measured in an exact way (values with uncertainties, for example). Many mathematical problems can be phrased as quantifier elimination problems (see [98, 187]).

The first real quantifier elimination procedure which has been implemented

was introduced by Collins in 1975 (see [36]). This method based on *cylindrical algebraic decomposition* (CAD) is worst-case doubly exponential in the number of variables. Some methods for solving the quantifier elimination problem have been proposed and implemented since the introduction of the CAD based algorithm (see QEPCAD[37], REDLOG[54], and QERRC[53]). A good survey of these methods can be found in [55].

7.2.1 Quantifier Elimination Problem

From a more formal point of view, the real *quantifier elimination problem* can be phrased as follows:

- *Given a formula F with quantified variables (universally $\forall$ and/or existentially $\exists$), find a formula $\overline{F}$ in which no variables are quantified, and that both F and $\overline{F}$ are equivalent in the domain of the real numbers.*

F is called the input formula, and $\overline{F}$ the solution formula. For example, one solution formula for $F : (\exists x)[ax^2 + bx + c = 0]$ can be:

$$\overline{F} : b^2 - 4ac \geq 0 \ \wedge \ [b \neq 0 \ \vee \ a \neq 0 \ \vee \ c = 0]$$

In the case of distance constraints, a formula $F : (\exists r \in [1, 2])[x^2 + y^2 = r^2]$ is equivalent to the following quantifier-free formula:

$$\overline{F} : (x^2 + y^2 \geq 1) \wedge (x^2 + y^2 \leq 4)$$

Using the QEPCAD implementation of the quantifier elimination algorithm (available in [24]), it is possible to transform a 2D quantified distance constraint into a set of non-quantified constraints in only some seconds, but in the general case, the 3D quantified distance constraint cannot be transformed[1].

In the next sections we show that it is possible to transform both types of constraints into a set of non-quantified constraints in less than one second, using a Specific Quantifier Elimination (SQE) algorithm based on graphic consideration. Moreover, this set of non-quantified constraints can be evaluated with interval arithmetics in order to build an inner box test.

[1] We use a PentiumIV 3GHz based machine with 512MB RAM and 2GB of swap memory. A general 2D quantified distance constraint is transformed into a quantifier-free formula in 33s, but the calculus for a general 3D quantified distance constraint could not be ended before memory overflow.

7.3 A Test Based on Quantifier Elimination

In this section, we propose a Specific Quantifier Elimination (SQE) algorithm for distance constraints $c_{\mathbf{a},\mathbf{r}}(x)$ in a two (and three) dimensional space. Higher dimensions are out of the scope of this algorithm[2].

Let $\rho_{\mathbf{a},\mathbf{r}}$ be the set of $x \in \mathbb{R}^n$ which verifies $c_{\mathbf{a},\mathbf{r}}(x)$ (see Figure 7.1 for an example in a two dimensional space). The main idea is to decribe $\rho_{\mathbf{a},\mathbf{r}}$ using a set of equations/inequalities depending only on the variable x. In other words, we transform a quantified distance constraint into an equivalent non quantified conjunction/disjunction of constraints which can be evaluated using classic interval arithmetics.

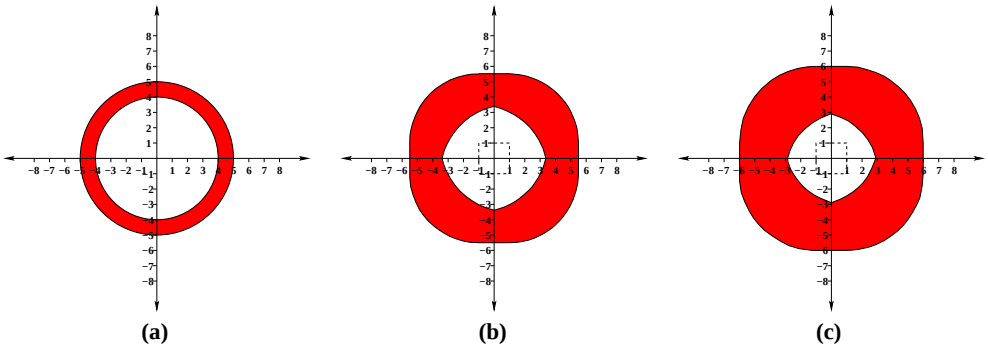


Figure 7.1: Some examples of quantified distance constraints and their solutions. (a) Only the parameter r has an interval value: $c_{a,\mathbf{r}}(x)$ with $a = (0, 0)$ and $\mathbf{r} = [4, 5]$. In (b) only the parameter a has an interval value: $c_{\mathbf{a},r}(x)$ with $\mathbf{a} = ([-1, 1][-1, 1])$ and $r = 4.5$. In (c) both parameters have interval values: $c_{\mathbf{a},\mathbf{r}}(x)$ with $\mathbf{a} = ([-1, 1], [-1, 1])$ and $\mathbf{r} = [4, 5]$.

7.3.1 The Two Dimensional Case

Let us consider a quantified distance constraint $c_{\mathbf{a},\mathbf{r}}(x)$ as following:

$$c_{\mathbf{a},\mathbf{r}}(x) : (\exists a \in \mathbf{a})(\exists r \in \mathbf{r})(f(a, x) = r^2) \tag{7.2}$$

where $a = (a_1, a_2)$, $x = (x_1, x_2)$, and $f(a, x) = (x_1 - a_1)^2 + (x_2 - a_2)^2$. Figure 7.2(a) graphically shows the set of x which verify this constraint.

[2]The main reason is that, because the limitation of the general QE algorithm, we perfom a hand-craft specific QE based on graphic consideration.

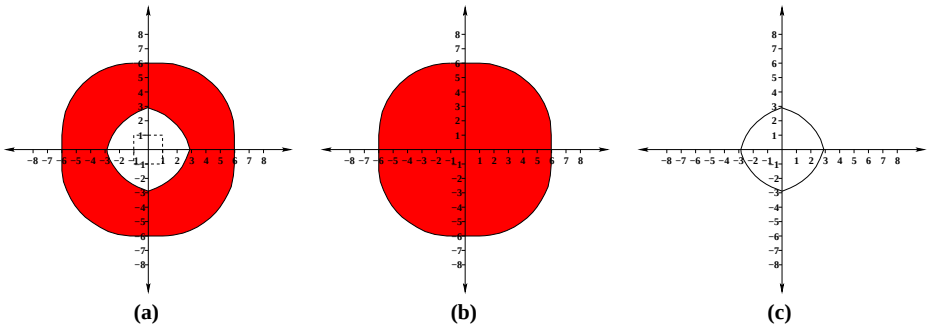


Figure 7.2: A two dimensional quantified distance constraint. (a) Set of solutions for the generic constraint $c_{\mathbf{a},\mathbf{r}}(x)$. (b) Representation of the internal auxiliary constraint $c^i_{\mathbf{a},\mathbf{r}}(x)$. (c) Representation of the external auxiliary constraint $c^e_{\mathbf{a},\mathbf{r}}(x)$.

The first step of the algorithm is the decomposition of $c_{\mathbf{a},\mathbf{r}}(x)$ into two auxiliary constraints. These constraints are the result of the elimination of the quantified parameter $\mathbf{r}$ from the constraint:

$$(\exists r \in \mathbf{r})(f(a,x) = r^2) \iff f(a,x) \leq \overline{r}^2 \wedge f(a,x) \geq \underline{r}^2 \tag{7.3}$$

Figure 7.2(b) and figure 7.2(c) show both auxiliary constraints. Let us call figure 7.2(b) the *internal auxiliary constraint* $c^i_{\mathbf{a},\mathbf{r}}(x)$ and figure 7.2(c) the *external auxiliary constraint* $c^e_{\mathbf{a},\mathbf{r}}(x)$. Therefore, $c_{\mathbf{a},\mathbf{r}}(x) \iff c^i_{\mathbf{a},\mathbf{r}}(x) \wedge \neg c^e_{\mathbf{a},\mathbf{r}}(x)$. These two auxiliary constraints can be characterized using the bounds of the involved interval, as presented below.

The constraint $c^i_{\mathbf{a},\mathbf{r}}(x)$

Using only the bounds of the intervals in $\mathbf{a}$ we obtain a disjunction of four non-quantified constraints that represent an approximation of $c^i_{\mathbf{a},\mathbf{r}}(x)$:

$$\begin{array}{ll} (x_1 - \underline{a}_1)^2 + (x_2 - \underline{a}_2)^2 \leq \overline{r}^2 & (x_1 - \underline{a}_1)^2 + (x_2 - \overline{a}_2)^2 \leq \overline{r}^2 \\ (x_1 - \overline{a}_1)^2 + (x_2 - \underline{a}_2)^2 \leq \overline{r}^2 & (x_1 - \overline{a}_1)^2 + (x_2 - \overline{a}_2)^2 \leq \overline{r}^2 \end{array} \tag{7.4}$$

These constraints do not describe the constraint $c^i_{\mathbf{a},\mathbf{r}}(x)$, because some gaps are still present (see Figure 7.3(a)). In order to fill the remaining gaps, two inclusion constraints are used (as shown in figure 7.3(b)) that are characterized in a compact way as follows:

$$x \in ([\underline{a}_1 - \overline{r}, \overline{a}_1 + \overline{r}], \mathbf{a}_2) \qquad x \in (\mathbf{a}_1, [\underline{a}_2 - \overline{r}, \overline{a}_2 + \overline{r}]) \tag{7.5}$$

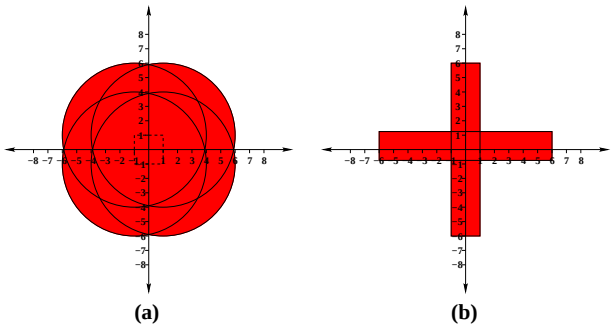


Figure 7.3: Reconstruction of $c^i_{\mathbf{a},\mathbf{r}}(x)$ using four disks (a) and two boxes (b).

Finally, $c^i_{\mathbf{a},\mathbf{r}}(x)$ is equivalent to the disjunction of the six non-quantified constraints presented in (7.4) and (7.5).

The constraint $c^e_{\mathbf{a},\mathbf{r}}(x)$

The graph of the constraint $c^e_{\mathbf{a},\mathbf{r}}(x)$ is easily obtained by intersecting four open disks. This constraint is represented as a conjunction of the following no quantified constraints:

$$\begin{array}{ll} (x_1 - \underline{a}_1)^2 + (x_2 - \underline{a}_2)^2 < \underline{r}^2 & (x_1 - \underline{a}_1)^2 + (x_2 - \overline{a}_2)^2 < \underline{r}^2 \\ (x_1 - \overline{a}_1)^2 + (x_2 - \underline{a}_2)^2 < \underline{r}^2 & (x_1 - \overline{a}_1)^2 + (x_2 - \overline{a}_2)^2 < \underline{r}^2 \end{array} \tag{7.6}$$

Notice that the boundary of the figure 7.2(c) is not included in the graph of $c^e_{\mathbf{a},\mathbf{r}}(x)$. The constraint $\neg c^e_{\mathbf{a},\mathbf{r}}(x)$ is then represented as the disjunction of four (non-strict) inequalities.

7.3.2 The Three Dimensional Case

In the three dimensional case, we use the same decomposition into two auxiliary constraints used in (7.3). The main difference is in the construction of the *internal auxiliary constraint*.

Consider the three dimensional distance constraint $c_{\mathbf{a},\mathbf{r}}(x)$ as in (7.2) but with $a = (a_1, a_2, a_3)$, $x = (x_1, x_2, x_3)$, and $f(a, x) = (x_1 - a_1)^2 + (x_2 - a_2)^2 + (x_3 - a_3)^2$. The generic graph of the constraint $c_{\mathbf{a},\mathbf{r}}(x)$ is shown in Figure 7.4.

As in the two dimensional case, the constraint $c_{\mathbf{a},\mathbf{r}}(x)$ is replaced by the conjunction of two auxiliary constraint $c^i_{\mathbf{a},\mathbf{r}}(x)$ and $\neg c^e_{\mathbf{a},\mathbf{r}}(x)$ (as shown in Figure 7.5).

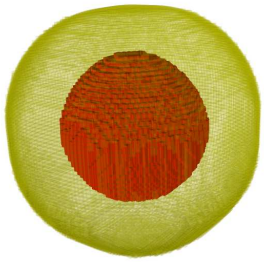


Figure 7.4: Generic graph of the constraint $c_{\mathbf{a},\mathbf{r}}(x)$. All values between the clearest surface and the darkest one are solutions of the constraint. Values inside the darkest surface are not solution.

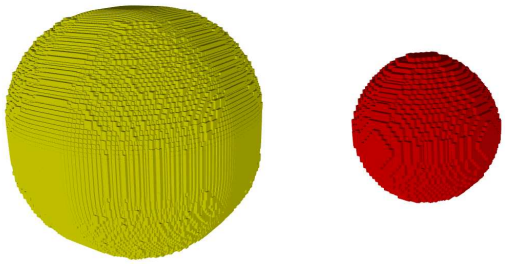


Figure 7.5: Decomposition of the constraint $c_{\mathbf{a},\mathbf{r}}(x)$ into two auxiliary constraints: $c^i_{\mathbf{a},\mathbf{r}}(x)$ (left-side picture) and $c^e_{\mathbf{a},\mathbf{r}}(x)$ (right-side picture).

The constraint $c^i_{\mathbf{a},\mathbf{r}}(x)$

Using the bounds of the intervals in $\mathbf{a}$ we build eight inequalities. The disjunction of these inequalities are a first approximation of $c^i_{\mathbf{a},\mathbf{r}}(x)$:

$$\begin{array}{l}
(x_1 - \underline{a}_1)^2 + (x_2 - \underline{a}_2)^2 + (x_3 - \underline{a}_3)^2 \leq \overline{r}^2 \\
(x_1 - \underline{a}_1)^2 + (x_2 - \underline{a}_2)^2 + (x_3 - \overline{a}_3)^2 \leq \overline{r}^2 \\
(x_1 - \underline{a}_1)^2 + (x_2 - \overline{a}_2)^2 + (x_3 - \underline{a}_3)^2 \leq \overline{r}^2 \\
(x_1 - \underline{a}_1)^2 + (x_2 - \overline{a}_2)^2 + (x_3 - \overline{a}_3)^2 \leq \overline{r}^2 \\
(x_1 - \overline{a}_1)^2 + (x_2 - \underline{a}_2)^2 + (x_3 - \underline{a}_3)^2 \leq \overline{r}^2 \\
(x_1 - \overline{a}_1)^2 + (x_2 - \underline{a}_2)^2 + (x_3 - \overline{a}_3)^2 \leq \overline{r}^2 \\
(x_1 - \overline{a}_1)^2 + (x_2 - \overline{a}_2)^2 + (x_3 - \underline{a}_3)^2 \leq \overline{r}^2 \\
(x_1 - \overline{a}_1)^2 + (x_2 - \overline{a}_2)^2 + (x_3 - \overline{a}_3)^2 \leq \overline{r}^2
\end{array} \tag{7.7}$$

As some gaps are still present in the graph of this approximation, a new set of constraints (based on geometric entities like boxes and cylinders) are also considered. Figure 7.6 shows the graph of three boxes characterized by the following

interval inclusion constraints:

$$\begin{array}{ll} x \in ([\underline{a}_1 - \overline{r}, \overline{a}_1 + \overline{r}], \mathbf{a}_2, \mathbf{a}_3) & x \in (\mathbf{a}_1, [\underline{a}_2 - \overline{r}, \overline{a}_2 + \overline{r}], \mathbf{a}_3) \\ x \in (\mathbf{a}_1, \mathbf{a}_2, [\underline{a}_3 - \overline{r}, \overline{a}_3 + \overline{r}]) & \end{array} \tag{7.8}$$

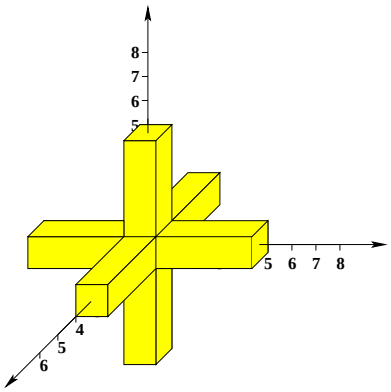


Figure 7.6: Graph of some interval inclusion constraints for building $c^i_{\mathbf{a},\mathbf{r}}(x)$.

The remaining gaps are filled with the following twelve cylindrical constraints (that correspond to the edges of the graph of $c^i_{\mathbf{a},\mathbf{r}}(x)$):

$$\begin{array}{l} (x_1 \in \mathbf{a}_1) \wedge ((x_2 - \underline{a}_2)^2 + (x_3 - \underline{a}_3)^2 \leq \overline{r}^2) \\ (x_1 \in \mathbf{a}_1) \wedge ((x_2 - \underline{a}_2)^2 + (x_3 - \overline{a}_3)^2 \leq \overline{r}^2) \\ (x_1 \in \mathbf{a}_1) \wedge ((x_2 - \overline{a}_2)^2 + (x_3 - \underline{a}_3)^2 \leq \overline{r}^2) \\ (x_1 \in \mathbf{a}_1) \wedge ((x_2 - \overline{a}_2)^2 + (x_3 - \overline{a}_3)^2 \leq \overline{r}^2) \\ (x_2 \in \mathbf{a}_2) \wedge ((x_1 - \underline{a}_1)^2 + (x_3 - \underline{a}_3)^2 \leq \overline{r}^2) \\ (x_2 \in \mathbf{a}_2) \wedge ((x_1 - \underline{a}_1)^2 + (x_3 - \overline{a}_3)^2 \leq \overline{r}^2) \\ (x_2 \in \mathbf{a}_2) \wedge ((x_1 - \overline{a}_1)^2 + (x_3 - \underline{a}_3)^2 \leq \overline{r}^2) \\ (x_2 \in \mathbf{a}_2) \wedge ((x_1 - \overline{a}_1)^2 + (x_3 - \overline{a}_3)^2 \leq \overline{r}^2) \\ (x_3 \in \mathbf{a}_3) \wedge ((x_1 - \underline{a}_1)^2 + (x_2 - \underline{a}_2)^2 \leq \overline{r}^2) \\ (x_3 \in \mathbf{a}_3) \wedge ((x_1 - \underline{a}_1)^2 + (x_2 - \overline{a}_2)^2 \leq \overline{r}^2) \\ (x_3 \in \mathbf{a}_3) \wedge ((x_1 - \overline{a}_1)^2 + (x_2 - \underline{a}_2)^2 \leq \overline{r}^2) \\ (x_3 \in \mathbf{a}_3) \wedge ((x_1 - \overline{a}_1)^2 + (x_2 - \overline{a}_2)^2 \leq \overline{r}^2) \end{array} \tag{7.9}$$

Finally, the graph of the constraint $c^i_{\mathbf{a},\mathbf{r}}(x)$ is equivalent to the disjunction of the twenty-three non-quantified constraints given the (7.7), (7.8), and (7.9).

The constraint $c^e_{\mathbf{a},\mathbf{r}}(x)$

As in the two dimensional case, the graph of the constraint $c^e_{\mathbf{a},\mathbf{r}}(x)$ is easily obtained by intersecting the following eight inequalities:

$$\begin{array}{l}
(x_1-\underline{a}_1)^2+(x_2-\underline{a}_2)^2+(x_3-\underline{a}_3)^2<\underline{r}^2\\
(x_1-\underline{a}_1)^2+(x_2-\underline{a}_2)^2+(x_3-\overline{a}_3)^2<\underline{r}^2\\
(x_1-\underline{a}_1)^2+(x_2-\overline{a}_2)^2+(x_3-\underline{a}_3)^2<\underline{r}^2\\
(x_1-\underline{a}_1)^2+(x_2-\overline{a}_2)^2+(x_3-\overline{a}_3)^2<\underline{r}^2\\
(x_1-\overline{a}_1)^2+(x_2-\underline{a}_2)^2+(x_3-\underline{a}_3)^2<\underline{r}^2\\
(x_1-\overline{a}_1)^2+(x_2-\underline{a}_2)^2+(x_3-\overline{a}_3)^2<\underline{r}^2\\
(x_1-\overline{a}_1)^2+(x_2-\overline{a}_2)^2+(x_3-\underline{a}_3)^2<\underline{r}^2\\
(x_1-\overline{a}_1)^2+(x_2-\overline{a}_2)^2+(x_3-\overline{a}_3)^2<\underline{r}^2
\end{array} \tag{7.10}$$

Note that the boundary is not included in the graph of $c^e_{\mathbf{a},\mathbf{r}}(x)$. The constraint $\neg c^e_{\mathbf{a},\mathbf{r}}(x)$ is then represented as the disjunction of eight (non-strict) inequalities.

7.3.3 Implementation

It is possible to represent a constraint in the form of a tree, where the internal nodes have logic operators *and* and *or*, and terminal nodes have non-quantified constraints or interval inclusion constraints. Figure 7.7 shows the generic tree of a quantified distance constraint $(x_1-\mathbf{a}_1)^2+(x_2-\mathbf{a}_2)^2=\mathbf{r}^2$.

Given an interval vector $\mathbf{x}$ and a decomposition tree, the algorithm computes an interval evaluation of the left side of each constraint and compares this evaluation with the right side. Each node of the tree has then a label (*true* or *false*), and the whole tree can be logically evaluated. If it is true then $\mathbf{x}$ is an inner box.

Some Optimizations

Based on the number of intervals in $\mathbf{a}$, it is possible to perform some optimization of the decomposition process. For example, in a two dimensional case, if only $\mathbf{a}_1$ and $\mathbf{r}$ are intervals and a_2 is a real value, the constraint $c^i_{\mathbf{a},\mathbf{r}}(x)$ is characterized by the disjunction of the following three constraints:

$$(x_1-\underline{a}_1)^2+(x_2-a_2)^2\leq\overline{r}^2 \quad (x_1-\overline{a}_1)^2+(x_2-a_2)^2\leq\overline{r}^2 \quad x\in(\mathbf{a}_1,[a_2-\overline{r},a_2+\overline{r}])$$

Moreover, if $(2\underline{r}\leq\overline{a}_1-\underline{a}_1)$, then the constraint $c^e_{\mathbf{a},\mathbf{r}}(x)$ is not considered (because the graph of $c_{\mathbf{a},\mathbf{r}}(x)$ has no gap in the middle), and the decomposition of

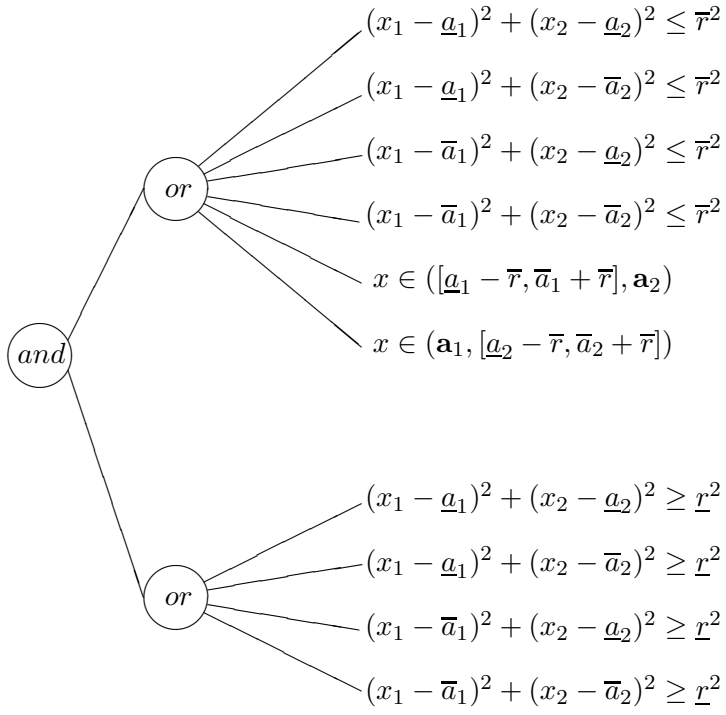


Figure 7.7: Generic decomposition tree of the constraint $c_{\mathbf{a},\mathbf{r}}(x)$.

$c_{\mathbf{a},\mathbf{r}}(x)$ is built as a disjunction of the last three inequalities.

The evaluation of the tree can be also optimized. It is performed with a Depth-First Search algorithm and the logic value of the intermediate *OR* nodes is computed inmediately when one of their child nodes get a *true* value.

7.3.4 Preliminary Results

This section presents different cases of quantified distance constraints (QDC) in 2D (see Table 7.1) and 3D (see Table 7.2), and the results of applying a classic solver[3] (without inner box test), a solver with a basic test (as that presented in Section 5.2.1 for the basic approach), and a solver combining an inner box test based on the Specific Quantifier Elimination (SQE) algorithm.

The first three constraints of Table 7.1 correspond to the examples presented in Figure 7.1 (Section 7.3). Constraint $c^{(4)}_{\mathbf{a},\mathbf{r}}(x)$ is a special example where the graph of the solution set has no gap in the middle.

[3] An implementation of a Branch and Prune algorithm for solving NCSPs (see Section 4.6.1).

QDC	Initial Domain	Parameters
$c^{(1)}_{a,\mathbf{r}}(x)$	$([-10,10],[-10,10])$	$a = (0,0)$ $\mathbf{r} = [4,5]$
$c^{(2)}_{\mathbf{a},r}(x)$	$([-10,10],[-10,10])$	$\mathbf{a} = ([-1,1],[-1,1])$ $r = 4.5$
$c^{(3)}_{\mathbf{a},\mathbf{r}}(x)$	$([-10,10],[-10,10])$	$\mathbf{a} = ([-1,1],[-1,1])$ $\mathbf{r} = [4,5]$
$c^{(4)}_{\mathbf{a},\mathbf{r}}(x)$	$([-10,10],[-10,10])$	$\mathbf{a} = ([-2,2],[0,0])$ $\mathbf{r} = [2,5]$
$c^{(5)}_{\mathbf{a},\mathbf{r}}(x)$	$([-10,10],[-10,10])$	$\mathbf{a} = ([-0.1,0.1],[-0.1,0.1])$ $\mathbf{r} = [3,5]$

Table 7.1: Some examples of quantified distances constraints in 2D.

Finally, constraint $c^{(5)}_{\mathbf{a},\mathbf{r}}(x)$ is an example with a big uncertainty in the parameter $\mathbf{r}$ and a little uncertainty in the parameter $\mathbf{a}$. The aim of these examples is to show the performance of the detection in constraints with different degrees of uncertainty.

QDC	Initial Domain	Parameters
$c^{(6)}_{\mathbf{a},\mathbf{r}}(x)$	$([-10,10],[-10,10],[-10,10])$	$\mathbf{a} = ([-0.1,0.1],[-0.1,0.1],[-0.1,0.1])$ $\mathbf{r} = [4,5]$
$c^{(7)}_{\mathbf{a},\mathbf{r}}(x)$	$([-10,10],[-10,10],[-10,10])$	$\mathbf{a} = ([-2,2],[-2,2],[-2,2])$ $\mathbf{r} = [4.4,4.5]$
$c^{(8)}_{\mathbf{a},\mathbf{r}}(x)$	$([-10,10],[-10,10],[-10,10])$	$\mathbf{a} = ([-2,2],[-2,2],[-2,2])$ $\mathbf{r} = [3,6]$

Table 7.2: Some examples of quantified distances constraints in 3D.

A Branch and Prune based solver was used to compute a sharp outer approximation of the continuum of solutions of each constraint. The inner box test was applied each time the filtering phase failed in reducing the domain of the variables.

Table 7.3 summarizes the computing results[4] of the experimentations. Row *Time* presents the running time in seconds. Rows *Boxes* and *Inner* present the total number of boxes and the number of inner boxes found, respectively.

The results show that in any case the use of inner box tests drastically reduces

[4]Obtained on a Pentium IV 3GHz with 512MB of RAM and 1.5GB of swap memory, running IcosAlias v0.2b (a tool in development in the COPRIN project) on a Linux operating system.

Prec. $w = 2e^{-2}$	**Without Test**	**Basic Test**	**SQE Test**
$c^{(1)}_{a,\mathbf{r}}(x)$			
Time (s)	4.516	0.358	0.361
Boxes	107,629	7,820	7,820
Inner	–	3,839	3,839
$c^{(2)}_{\mathbf{a},r}(x)$			
Time (s)	21.946	22.783	0.505
Boxes	510,307	510,306	9,540
Inner	–	–	4,551
$c^{(3)}_{\mathbf{a},\mathbf{r}}(x)$			
Time (s)	31.640	33.151	0.515
Boxes	754,768	754,768	9,559
Inner	–	–	4,556
$c^{(4)}_{\mathbf{a},\mathbf{r}}(x)$			
Time (s)	19.767	18.042	0.290
Boxes	470,776	410,412	5,896
Inner	–	1,328	2,776
$c^{(5)}_{\mathbf{a},\mathbf{r}}(x)$			
Time (s)	4.407	2.293	0.411
Boxes	170,214	50,338	7,147
Inner	–	2,604	3,590
Prec. $w = 2e^{-1}$	**Without Test**	**Basic Test**	**SQE Test**
$c^{(6)}_{\mathbf{a},\mathbf{r}}(x)$			
Time (s)	6.754	5.469	4.420
Boxes	124,832	97,384	48,458
Inner	–	12,616	15,218
$c^{(7)}_{\mathbf{a},\mathbf{r}}(x)$			
Time (s)	58.767	62.634	9.805
Boxes	1,162,876	1,162,876	99,827
Inner	–	–	25,467
$c^{(8)}_{\mathbf{a},\mathbf{r}}(x)$			
Time (s)	79.263	84.294	13.288
Boxes	1,592,076	1,592,076	147,648
Inner	–	–	37,996

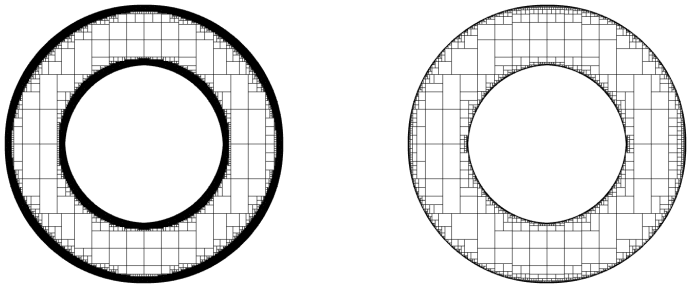
Table 7.3: Computed results of the experiments. All constraints in 2D are computed with a precision $\epsilon = 2e^{-2}$. Precision for constraints in 3D was $\epsilon = 2e^{-1}$.

the computing time. It is normal because the set of boxes that have proved to be inside the continua of solutions are no more split. Table 7.3 shows that the

test based on the SQE is the best in most of cases but in constraint $c^{(1)}_{a,\mathbf{r}}(x)$. The reason is that for this constraint, the basic inner box test is optimal (see Section 7.1), so the cost of the decomposition may lightly increase the computing time.

It is important to note that the basic inner box test detected inner boxes only in the constraints $c^{(1)}_{a,\mathbf{r}}(x)$, $c^{(4)}_{\mathbf{a},\mathbf{r}}(x)$, $c^{(5)}_{\mathbf{a},\mathbf{r}}(x)$, and $c^{(6)}_{\mathbf{a},\mathbf{r}}(x)$. In most of these cases, the degree of the uncertainties in the center parameter $\mathbf{a}$ is low (with respect to the degree of the distance parameter $\mathbf{r}$), and the test can detect a few number of boxes.

Figure 7.8 shows a graphic comparison between the solution set computed with the basic inner box test (left-side picture) and the results obtained with the test based on the SQE (right-side picture).



(a) **(b)**

Figure 7.8: The computed solution set of the constraint $c^{(5)}_{\mathbf{a},\mathbf{r}}(x)$. (a) Using the basic inner box test, (b) Using the test based on SQE.

In the first case, the test cannot detect inner boxes in a big surface (due to the existential parameters in the left-side of the constraint), and therefore, the solver splits the boxes inside this zone until arriving to the given precision.

On the other hand, the test based on the SQE successfully detects the inner boxes inside this surface, and the number of boxes needed for describing the continua of solutions is substantially lower. Similar results can be observed in the three dimensional cases, but the performance of the inner box test is lower.

It is important to note that both tests are only *sufficient* conditions (and they are not necessary ones). In other words, a box can be inside the continuum of solutions while it does not satisfy any of the constraints generated by the decomposition. Such a box would intersect several graphs of the generated constraints but would be included in none of them. This situation is called the *decomposition flaw* and it is generally found in the 3D decomposition.

Another limitation of the test based on the SQE is the *scalability*. It has been presented for constraints in two and three dimensional spaces, but higher dimensions are out of the scope of the decomposition.

In orden to overcome some of these limitations, the next section studies another approach for detecting inner boxes. This approach is based on the Generalized Intervals and a result of Modal Interval Analysis.

7.4 A Test based on Generalized Interval

In this section a sufficient condition is proposed for a n-dimensional box $\mathbf{x}$ to be an inner box. It is based on one evaluation of the expression of $f(x, a)$ using generalized intervals and their arithmetic. This technique is based on an important result of the Modal Interval theory (see [66, 71, 96]) roughly explained in Section 3.7.3.

Let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be a continuous function and $\mathbf{x} \in \mathbb{KR}^n$ be a generalized interval vector. Let $\mathbf{x}_P$ be a generalized interval vector containing all the proper interval components of $\mathbf{x}$, and let $\mathbf{x}_I$ be another generalized interval vector containing all the improper ones. As shown in Example 3.7.4, a generalized interval $\mathbf{z}$ will be $(f, \mathbf{x})$-interpretable if and only if

$$(\forall x_P \in \mathbf{x}_P)(Qz \in Pro(\mathbf{z}))(\exists x_I \in Pro(\mathbf{x}_I))(f(x) = z) \tag{7.11}$$

where $Q = \forall$ if $\mathbf{z}$ is an improper interval and $Q = \exists$ if $\mathbf{z}$ is a proper one. Notice that it is exactly the semantic needed for the detection of inner boxes in distance constraints. The problem is now, to compute such interval $\mathbf{z}$.

As it has been noted in [66, 71], under some conditions (notably the single-occurrence of the variables with improper intervals associated), it is possible to compute such a generalized interval by using Kaucher arithmetic to evaluate the given function. Although it was not introduced with this goal, the Kaucher operation $\mathbf{x} \diamond \mathbf{y}$, where $\diamond \in \{+, -, \times, /\}$, is proved to raise the minimal[5] $(\diamond, \mathbf{x}, \mathbf{y})$-interpretable generalized intervals.

As univariate functions $f(x)$ like x^2 or $\sqrt{x}$ are also extended to generalized intervals (by computing the interval $\mathbf{z} = f(Pro(\mathbf{x}))$ and keeping the same proper/improper quality), it is possible to provide interpretable intervals for more realistic functions compounded of elementary functions.

[5] In the sense of the generalized interval inclusion.

7.4.1 Generalized Interval Evaluation of a Distance Constraint

Consider a quantified distance constraint $c_{\mathbf{a},\mathbf{r}}(x)$, between two points x and a in a n-dimensional space, defined as follows:

$$c_{\mathbf{a},\mathbf{r}}(x) : (\exists a \in \mathbf{a})(\exists r \in \mathbf{r})(f(a,x) = r^2) \tag{7.12}$$

where $a = (a_1, \ldots, a_n)$, $x = (x_1, \ldots, x_n)$, and $f(a,x) = \sum_{i=1}^{n}(x_i - a_i)^2$. Notice that in these types of distance constraints, each existentially quantified parameter has only one occurrence. Therefore, using Kaucher arithmetic it is possible to compute a $(f, \mathbf{x}, Dual(\mathbf{a}))$-interpretable generalized intervals by evaluating:

- $\mathbf{z}_i = \mathbf{x}_i - Dual(\mathbf{a}_i)$,
- $\mathbf{z}'_i = \mathbf{z}_i^2$,
- $\mathbf{z} = \sum_{i=1}^{n} \mathbf{z}'_i$.

As a consequence, the generalized interval evaluation $\mathbf{z} = f(Dual(\mathbf{a}), \mathbf{x})$ given by this process can be semantically interpreted as follows:

$$(\forall x \in \mathbf{x})(Qz \in \mathbf{z})(\exists a \in \mathbf{a})(f(a,x) = z) \tag{7.13}$$

Furthermore, thanks to the properties of the generalized intervals inclusion (see Section 3.7.1), if $\mathbf{z} \subseteq \mathbf{r}^2$ then $\mathbf{r}^2$ is also $(f, \mathbf{x}, Dual(\mathbf{a}))$-interpretable, that is

$$(\forall x \in \mathbf{x})(\exists r \in \mathbf{r})(\exists a \in \mathbf{a})(f(a,x) = r^2) \tag{7.14}$$

which is exactly the semantics of the quantified distance constraint.

Finally, the inclusion $f(Dual(\mathbf{a}), \mathbf{x}) \subseteq \mathbf{r}^2$ is a sufficient condition for detecting an inner box $\mathbf{x}$. Notice that this condition is not necessary in general. For example, consider $\mathbf{a} = ([-1,1],[-1,1])$ and $\mathbf{r} = [1,1]$ so that $\mathbf{x} = ([-1,1],[-1,1])$ is an inner box which does not satisfy $f(Dual(\mathbf{a}), \mathbf{x}) \subseteq \mathbf{r}$ (in this case, the test based on SQE presented in Section 7.3 succeeds in proving it). However, it can be proved that this sufficient condition is furthermore necessary provided that $m(\mathbf{a}_i) \notin \mathbf{x}_i$ $(\forall i = 1..n)$, which is likely to be met for inner boxes $\mathbf{x}$ in some realistic situations.

7.5 Inner Boxes for Systems of Distance Equations

The tests discussed in sections 7.3 and 7.4 were designed to detect inner boxes in only one quantified distance constraint. Of course, the solving process of a system of distances constraints with uncertainties needs to detect boxes included inside the continuum of solutions of the whole system, and not only inside a single constraint.

It can be noted that a sufficient condition designed for one quantified distance constraint can also be used for a conjunction of quantified distance constraints:

$$C = \{c^{(1)}_{\mathbf{a}^{(1)},\mathbf{r}^{(1)}}(x), \ldots, c^{(n)}_{\mathbf{a}^{(n)},\mathbf{r}^{(n)}}(x)\} \tag{7.15}$$

where each $\mathbf{a}^{(k)}$ is a n-dimensional box, and each $\mathbf{r}^{(k)}$ is an interval. Indeed, if existentially quantified parameters are not shared between different constraints[6], we have the following implication:

$$\bigwedge_{k=1}^{n} (\mathbf{x} \subseteq \rho_{c^{(k)}}) \Longrightarrow \mathbf{x} \subseteq \rho_C \tag{7.16}$$

where $\rho_{c^{(k)}}$ is the set of solutions of the constraint $c^{(k)}_{\mathbf{a}^{(k)},\mathbf{r}^{(k)}}(x)$ and ρ_C represents the set of solutions of the problem. In other words, if the box $\mathbf{x}$ is an inner box for each constraint of the problem then it is an inner box for the whole problem.

Notice that it is true only if existentially quantified parameters are not shared between different constraints, because the existence of a value $a \in \mathbf{a}$ which satisfies the constraint $c^{(1)}$ and a value $a' \in \mathbf{a}$ which satisfies the constraint $c^{(2)}$ does not imply the existence of a value $a'' \in \mathbf{a}$ which satisfies $c^{(1)} \wedge c^{(2)}$ simultaneously.

7.5.1 Quantifier Elimination versus Generalized Interval

Some academic examples were selected in order to compare both approaches for checking inner boxes. Problem 1 and Problem 2 are in a two dimensional space, while Problem 3 is in a three dimensional space. The first problem is composed of a single constraint. The second and third problem are composed of three constraints. All problems have uncertainties. Table 7.4 shows the description of each one.

A Branch and Prune algorithm combining 2B-consistency and bisection techniques was used for solving each problem.

[6]Otherwise, other techniques have to be used (see [65, 71]).

Problem	Constraints	Domains
Problem 1	$c_{\mathbf{a},\mathbf{r}}(x)$	$\mathbf{x} = ([-100,100],[-100,100])$ $\mathbf{a} = ([-0.5,0.5],[-0.5,1.3])$ $\mathbf{r} = [1.3,1.6]$
Problem 2	$c_{\mathbf{a}^1,\mathbf{r}^1}(x)$ $c_{\mathbf{a}^2,\mathbf{r}^2}(x)$ $c_{\mathbf{a}^3,\mathbf{r}^3}(x)$	$\mathbf{x} = ([-100,100],[-100,100])$ $\mathbf{a}^1 = (0,0)$ $\mathbf{r}^1 = [2,2.25]$ $\mathbf{a}^2 = ([3,3.5],0)$ $\mathbf{r}^2 = [2.95,3.05]$ $\mathbf{a}^3 = ([-2.5,-2.25],2)$ $\mathbf{r}^3 = [3.25,3.5]$
Problem 3	$c_{\mathbf{a}^1,\mathbf{r}^1}(x)$ $c_{\mathbf{a}^2,\mathbf{r}^2}(x)$ $c_{\mathbf{a}^3,\mathbf{r}^3}(x)$	$\mathbf{x} = ([0,100],[-100,100],[0,100])$ $\mathbf{a}^1 = ([-0.1,0.1],[-0.1,0.1],[-0.1,0.1])$ $\mathbf{r}^1 = [4,5]$ $\mathbf{a}^2 = ([4.9,5.1],[-0.1,0.1],[-0.1,0.1])$ $\mathbf{r}^2 = [3,4]$ $\mathbf{a}^3 = ([1.8,2.2],[3.95,4.05],[0.8,1.2])$ $\mathbf{r}^3 = [4,5]$

Table 7.4: Description of some academic examples.

The inner box test was applied each time the consistency algorithm failed in reducing the domain of the variables. Table 7.5 shows the computational results[7] of the experiments, using the specific quantifier elimination (SQE Test) and the generalized interval evaluation (GIE Test).

Rows *Boxes* and *Inner* present the total number of boxes and the number of inner boxes found, respectively. Row *Volume* shows the total volume of the boxes, while row *IVolume* shows the volume of the inner boxes. Row *Time* presents the running time in seconds.

Some experiments have been conducted without using any inner box test (only for information), but Problem 1 led to swap memory overflow (1.6Go) before reaching the expected precision.

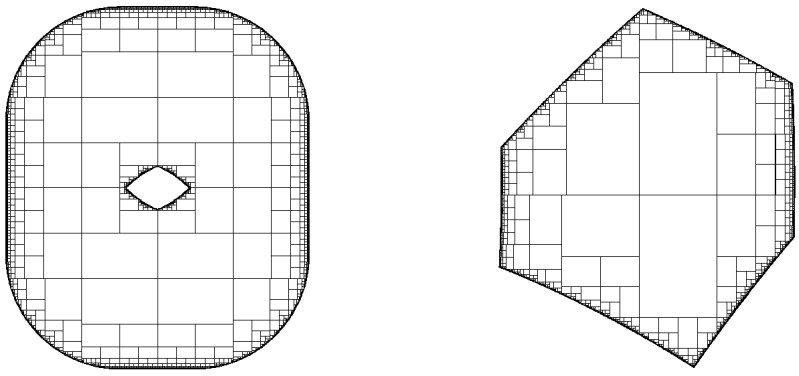
On Problem 1 and Problem 2 (Figure 7.9), both approaches are optimal and compute exactly the same approximations: on one hand, the bisection is performed in such a way that the *decomposition flaw* (Section 7.3.4) of the SQE is not met. On the other hand, we have $m(\mathbf{a}_i) \notin \mathbf{x}_i$ $(\forall i = 1..n)$ for all inner boxes, so that the GIE Test is optimal. The running time using the GIE Test is always slightly

[7]Obtained on a Pentium IV 2GHz with 256Mb of RAM and 1,5Gb of swap memory, running IcosAlias v0.2b (http://www-sop.inria.fr/coprin/gchabert/icosalias.html).

	No Test	SQE Test	GIE Test
Problem 1 ($\epsilon = 10^{-3}$)			
Boxes	$> 10^7$	64877	64877
Inner	–	33225	33225
Volume	–	18.50312	18.50312
IVolume	–	18.49187	18.49187
Time (sec.)	–	4.63	4.08
Problem 2 ($\epsilon = 10^{-3}$)			
Boxes	451655	5481	5481
Inner	–	2550	2550
Volume	0.21236	0.21236	0.21236
IVolume	–	0.21103	0.21103
Time (sec.)	36,08	0.53	0.43
Problem 3 ($\epsilon = 10^{-2}$)			
Boxes	7717507	503059	501795
Inner	–	137900	137799
Volume	2.83133	2.83133	2.83133
IVolume	–	2.72203	2.72254
Time (sec.)	803.63	87.49	58.38

Table 7.5: Computational results of the experiments.

lower than using the SQE Test because the former computes only one evaluation of the constraint.



(a) **(b)**

Figure 7.9: A graphic representation of the continua of solutions. (a) Solutions of Problem 1, (b) Solutions of Problem 2.

On Problem 3 (Figure 7.10), the two tests compute different approximations:

the total volumes are equal with both methods but the inner volume provided by the GIE Test is greater, with a lower number of inner boxes. While the GIE Test is still optimal (because $m(\mathbf{a}_i) \notin \mathbf{x}_i$ $(\forall i = 1..n)$), the *decomposition flaw* is now met (in dimension 3 the decomposition used for the SQE is more complicated so that the *decomposition flaw* is more likely to be met). As a consequence, the speedup of GIE Test is more sensitive on this example.

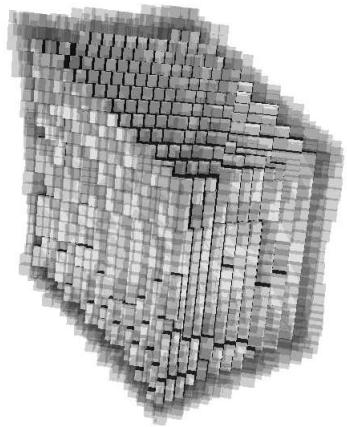


Figure 7.10: Graphic representation of the solutions for Problem 3.

7.6 An Optimal Inner Box Test for Distance Equations

This section introduces an optimal inner box test for distance constraints with existentially quantified parameters, based on generalized interval evaluation and geometric considerations.

As shown in the Section 7.4.1, the Modal Interval theory (see Section 3.7.3), guarantees that if $f(x, a)$ has a single occurrence of the variable[8] a, the evaluation of $\mathbf{z} = f(\mathbf{x}, Dual(\mathbf{a}))$ using generalized interval arithmetic produces a $(f, \mathbf{x}, Dual(\mathbf{a}))$-interpretable interval $\mathbf{z}$, but it does not say anything about the optimality of the computed interval.

Actually, for the four basic operations $\{+, -, \times, /\}$, the computed interval $\mathbf{z}$ is optimal. Even for the *elementary* function $sqr, sqrt, cos, sin, \ldots$ the resulting interval will be optimal, but it does not mean that the composition of functions will be optimal too. As counter-example, consider the expression $f(a, x) = (x - a)^2$

[8] Actually, a single occurrence of each component of the interval vector a, in a general case.

with $\mathbf{x} = [-1, 1]$ and $\mathbf{a} = [-1, 1]$. The result of evaluating $\mathbf{z} = f(\mathbf{x}, Dual(\mathbf{a}))$ involves two steps:

$$\mathbf{z}_{tmp} = (\mathbf{x} - Dual(\mathbf{a})) = ([-1, 1] - [1, -1]) = [0, 0] \quad (7.17)$$

$$\mathbf{z} = (\mathbf{z}_{tmp})^2 = ([0, 0])^2 = [0, 0] \quad (7.18)$$

The final result $\mathbf{z} = [0, 0]$ is $(f, \mathbf{x}, Dual(\mathbf{a}))$-interpretable but is not optimal[9], even though each performed operation is optimal (that means, each operation ϕ returns the minimal interpretable interval $\mathbf{z}$ (i.e. which verifies the $(\phi, \mathbf{x}, Dual(\mathbf{a}))$-interpretability).

This drawback basically occurs because the expression is split into sub-expressions that all are required to verify an *inner property* by itself, regardless of possible overlays due to non-monotonicity. This condition is stronger than it is needed.

Indeed, non monotonicity (e.g. of the square function) means that the same value (e.g., 4) may be obtained by several different values of the sub-expression $(x - a)$ (-2 and 2). If the value 2 of the sub-expression $(x - a)$ can be obtained for all values of a part of the box $\mathbf{x}$, and if -2 can be obtained for all values of the other part, then the value 4 for $(x - a)^2$ can be obtained for all values in $\mathbf{x}$. Such *overlay* cannot be detected by generalized interval arithmetic since neither 2 nor -2 appears to be obtained for all values in $\mathbf{x}$.

In the case of distance equations, we identify this situation in the form of a *symmetry* of the expression $(x - \mathbf{a})^2$ w.r.t. the middle point of $\mathbf{a}$ (see Figure 7.11).

Definition 18 (Symmetric Interval) *Let* $\mathbf{x} = [\underline{x}, \overline{x}] \in \mathbb{IR}$ *and* $\tilde{a} \in \mathbb{R}$. *The interval* $\mathbf{x}^* = [\underline{x}^*, \overline{x}^*] \in \mathbb{IR}$ *is a symmetric interval of* $\mathbf{x}$ *w.r.t.* $\tilde{a}$, *iff*

$$\frac{(\underline{x} + \overline{x}^*)}{2} = \frac{(\overline{x} + \underline{x}^*)}{2} = \tilde{a} \quad (7.19)$$

This definition allows us to formulate the following proposition:

Proposition 7 *Let* $\mathbf{x}, \mathbf{a} \in \mathbb{IR}$ *such that* $\mathbf{x} \cap \{m(\mathbf{a})\} \in \{\emptyset, \{\underline{x}\}, \{\overline{x}\}\}$ *and consider a function* $f(x, a) = (x - a)^2$. *The interval* $\mathbf{x}$ *has a symmetric interval* $\mathbf{x}^*$ *w.r.t. the middle point of* $\mathbf{a}$, *such that the evaluation* $f(\mathbf{x}, Dual(\mathbf{a})) = f(\mathbf{x}^*, Dual(\mathbf{a}))$.

Proof. Let $\tilde{a} = m(\mathbf{a})$ be the middle point of $\mathbf{a}$, and $\mathbf{x}^* = [2\tilde{a} - \overline{x}, 2\tilde{a} - \underline{x}]$ be the

[9]Note that the minimal interval $\mathbf{z}$ verifying the $(f, \mathbf{x}, Dual(\mathbf{a}))$-interpretability is $\mathbf{z} = [1, 0]$, but it cannot be obtained with a simple generalized interval evaluation of the expression.

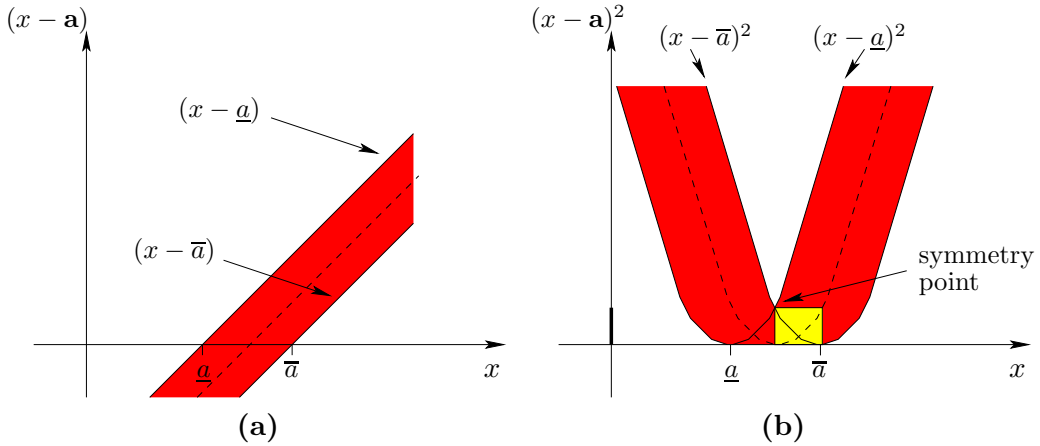


Figure 7.11: (a) Graphic representation of the expression $(x - \mathbf{a})$, without symmetry. (b) Graphic representation of $(x - \mathbf{a})^2$. There is a symmetry w.r.t. the middle point of the interval $\mathbf{a}$.

symmetric interval of $\mathbf{x} = [\underline{x}, \overline{x}]$ w.r.t. $\tilde{a}$. Then,

$$\begin{aligned}
f(\mathbf{x}^*, Dual(\mathbf{a})) &= ([2\tilde{a} - \overline{x}, 2\tilde{a} - \underline{x}] - [\overline{a}, \underline{a}])^2 \\
&= ([2\tilde{a} - \overline{x} - \underline{a}, 2\tilde{a} - \underline{x} - \overline{a}])^2 \\
&= ([2\tfrac{(\underline{a}+\overline{a})}{2} - \overline{x} - \underline{a}, 2\tfrac{(\underline{a}+\overline{a})}{2} - \underline{x} - \overline{a}])^2 \\
&= ([\underline{a} + \overline{a} - \overline{x} - \underline{a}, \underline{a} + \overline{a} - \underline{x} - \overline{a}])^2 \\
&= ([\overline{a} - \overline{x}, \underline{a} - \underline{x}])^2 \\
&= ([-(\overline{x} - \overline{a}), -(\underline{x} - \underline{a})])^2 \\
&= ((-1)[\underline{x} - \underline{a}, \overline{x} - \overline{a}])^2 \\
&= ([\underline{x}, \overline{x}] - [\overline{a}, \underline{a}])^2 \\
&= f(\mathbf{x}, Dual(\mathbf{a}))
\end{aligned}$$

□

Now, consider an interval $\mathbf{x} \in \mathbb{IR}$ such that $\mathbf{x} \cap \{m(\mathbf{a})\} \notin \{\emptyset, \underline{x}, \overline{x}\}$, and let $\mathbf{x}_{left} = [\underline{x}, m(\mathbf{a})]$ and $\mathbf{x}_{right} = [m(\mathbf{a}), \overline{x}]$ be two intervals generated by bisecting the initial interval $\mathbf{x}$ in the middle point of the interval $\mathbf{a}$. Because the symmetry (Proposition 7), we do not need to evaluate the expression $(x - Dual(\mathbf{a}))^2$ in the whole interval $\mathbf{x}$ but only in one of the intervals $\mathbf{x}_{left}$ or $\mathbf{x}_{right}$ (the biggest[10] one). The resulting interval $\mathbf{z} = (\mathbf{x}' - Dual(\mathbf{a}))^2$ (with $\mathbf{x}' \in \{\mathbf{x}_{left}, \mathbf{x}_{right}\}$) will not only be $(f, \mathbf{x}, Dual(\mathbf{a}))$-interpretable but also optimal.

[10]Because the evaluation of the smallest interval is already taken into account in the evaluation of the biggest one.

The optimality of this evaluation can be proved by studying its behavior in different parts of the domain. Without loss of generality, consider the interval $\mathbf{x}' = \mathbf{x}_{right}$ to be the biggest one.
The evaluation $\mathbf{z} = f(\mathbf{x}', Dual(\mathbf{a})) = ([\underline{x}' - \underline{a}, \overline{x}' - \overline{a}])^2$ involves two expressions:

- $(\underline{x}' - \underline{a})$, which is equal to $(m(\mathbf{a}) - \underline{a}) = \frac{\overline{a}-\underline{a}}{2}$, always greater or equal than zero (see Figure 7.11(b) in the symmetry point), and
- $(\overline{x}' - \overline{a})$, which depends on the value of $\overline{x}'$. If $(m(\mathbf{a}) \leq \overline{x}' \leq \overline{a})$ then the expression $(\overline{x}' - \overline{a})$ is such that $-\frac{\overline{a}-\underline{a}}{2} \leq (\overline{x}' - \overline{a}) \leq 0$, and $\mathbf{z} = [(\frac{\overline{a}-\underline{a}}{2})^2, 0]$ which is the optimal evaluation (see the yellow box of Figure 7.11(b)), else $(\overline{x}' > \overline{a})$ and then $(\overline{x}' - \overline{a}) > 0$. The evaluation will be $\mathbf{z} = [(\frac{\overline{a}-\underline{a}}{2})^2, (\overline{x}' - \overline{a})^2]$ which is again optimal.

Finally, in order to build an optimal inner box test for distance constraints with existentially quantified parameters, each term $(\mathbf{x}_i - \mathbf{a}_i)^2$ is evaluated according the following rule: *If* $\mathbf{x}_i \cap \{m(\mathbf{a}_i)\} \in \{\emptyset, \{\underline{x}\}, \{\overline{x}\}\}$ *then a usual generalized interval evaluation is performed to compute the value of* $\mathbf{z}_i$*, else the interval* $\mathbf{x}_i$ *is bisected into* $\mathbf{x}_{i_{left}} = [\underline{x_i}, m(\mathbf{a}_i)]$ *and* $\mathbf{x}_{i_{right}} = [m(\mathbf{a}_i), \overline{x_i}]$ *and* $\mathbf{z}_i = (\mathbf{x}'_i - \mathbf{a})^2$*, where* $\mathbf{x}'_i$ *is the biggest between* $\mathbf{x}_{i_{left}}$ *and* $\mathbf{x}_{i_{right}}$*.*

7.7 Conclusion

Equality constraints with existentially quantified parameters, i.e. constraints like $(\exists a \in \mathbf{a})(f(a, x) = 0)$, generally have continua of solutions. Therefore, any bisection algorithm dedicated to the approximation of their solutions should incorporate a test for checking if a box is included inside the continuum of solutions. Unless it will spend most of the time bisecting again and again boxes included in the solution set.

This chapter studied different tests for detecting inner boxes. First, it showed how interval arithmetics can be successfully used for detecting inner boxes in some types of constraints and why this arithmetics is less effective in a more general case of quantified constraint. In addition, the quantified elimination problem was presented within an explanation of how the solution of this problem can be used with classic interval arithmetics in order to improve the detection of inner boxes.

A special quantified elimination algorithm based on geometric considerations was also proposed. This algorithm allows one to transform a quantified distance constraint into a set of non-quantified constraints in less than one second. Classic

interval arithmetic is then used to detect boxes included inside a continuum of solutions.

On the other hand, using a new point of view, a test which was initially proposed by the modal intervals theory has been introduced. This new test combines generalized interval arithmetic and a semantic interpretation of the result to detect inner boxes in quantified constraints.

Some experiments have been conducted on academic examples of conjunctions of quantified distance constraints in a two- and three-dimensional space. Although both methods are very different, they raise very similar results about both computation times and description of the solution set (with a slight advantage for the test based on generalized intervals).

This situation drastically changes when arbitrary dimensions (greater than 3) are considered. While the test based on generalized interval evaluation can be trivially extended, the proposed test based on specific quantifier elimination fails. Moreover, the former is much simpler to implement.

Notice that both tests are only sufficient conditions. That means some boxes which are inside the continuum of solutions may not be detected[11]. For this reason, a deeper study of these tests has been performed. In the case of the SQE, it is difficult to overcome the *decomposition flaw*, because it is in the basis of the approach. In the case of the generalized interval evaluation (and using geometric consideration), we showed that an optimal inner box test can be built. This is the first test which can be considered as a necessary a sufficient condition for detecting inner boxes in the discussed quantified distance constraints. As well as being optimal, it is also cheap, because it only involves interval evaluations and some verifications that can be performed in a linear time (w.r.t the terms involved in the constraint). Moreover, it can be interesting to study its applicability to others types of constraints verifying a symmetry condition.

Anyway, after the study of the inner box test and the description of the continua of solutions using a set of boxes, an important drawback of this approach has been detected. In a two dimensional space, any interval based solver must describe the border of the continuum of solutions (a line) with small boxes (these boxes will have the given precision used as stop condition). In a three dimensional space, an interval based solver must describe a surface (the border of the continuum of solutions in the space) with the same precision. That means, in

[11] Anyway, it can be proved that after a finite number of bisections, all the generated boxes will be detected as inner ones.

a n-dimensional space any interval based solver will describe a (n-1)-dimensional volume with boxes smaller than the given precision, even though an optimal inner box test be applied.

This is a weakness of the interval based approach that motivates the study performed in the next chapter.

Chapter 8

Generalized Interval Projection

This chapter is based on the work presented in [76]. As discussed in the conclusions of Chapter 7, an important weakness of the use of an interval based solver (e.g. a Branch and Prune algorithm with a given precision) for describing continua of solutions of a NCSP, is that in a n-dimensional space this solver will describe a $(n-1)$-dimensional volume with a lot of boxes having the given precision (even though an optimal inner box test be applied).

In problems involving quantified constraints and several continua of solutions it can be interesting to have an inner-(rough)-approximation as the same as an outer-approximation. That means, to compute a small box which conservatively approximates a continuum of solutions and a big box fully included inside the same continuum.

In this chapter, a new technique for building such a box in a constraint with existentially quantified parameters is presented. This approach is able to build an inner box for the problem starting with a single point solution, by consistently extending the domain of every variable. The key point is a new method called *generalized projection* which is based on the extended algebraic structure provided by generalized intervals. The requirements are that each parameter must occur only once in the system, variable domains must be bounded, and each variable must occur only once in each constraint.

The outline of the chapter is as follows: Section 8.1 presents a brief introduction and a motivating example for the domains extension. Section 8.2 remembers the inner box test which will be used in Section 8.3 to perform consistent domains extension. Section 8.4 shows the application of the extension algorithm on the introductory problem. Finally, Section 8.5 presents the conclusions of the chapter.

8.1 Introduction

The purpose of the work presented in this chapter will be illustrated on a simple example of signal relay positioning. The situation is as follows: *m units are deployed on an area, each of them being equipped with a transceiver. Because of the limited transmission distance of their transceivers, the units cannot communicate. The question is to find a good position for a relay such that all units get connected (can communicate between them).*

Denote (a_i, b_i) the coordinates of the i^{th} unit position, and d_i its distance from the relay. Assume first that all a_i, b_i and d_i are fixed. Then, the model consists in m simple distance equations and can be solved by any traditional algebraic or numerical technique presented in Section 4.7. Since the system is probably unfeasible, a least-square method can provide a point making each distance being as close as possible to the desired value d_i.

Unfortunately, this model suffers from three serious limitations:

1. Distances should not be fixed. The distance d_i must be neither more than the transceiver range $\overline{\mathbf{d}_i}$, nor less than a lower bound $\underline{\mathbf{d}_i}$, say, because of the damaging loop effect. Hence, distances must rather be assigned intervals $\{\mathbf{d}_1,\ldots,\mathbf{d}_m\}$.

2. Positions of units are not fixed neither. They usually patrol around their position and can move in a zone (box $(\mathbf{a}_i, \mathbf{b}_i)$) to pick up the signal.

3. Providing a single solution (x, y) is often not realistic. For example, an antenna cannot be installed exactly at a precise position in presence of obstacles. Therefore, one is rather interested by a box $(\mathbf{x}, \mathbf{y})$ such that any position chosen in this box is appropriate. Obviously, the wider the box, the better.

Considering the above limitations, a better model for the problem can be defined considering a set of m quantified distance constraints $c^{(i)}(x, y)$ as follows:

$$c^{(i)}(x,y) : \exists (a_i, b_i, d_i) \in (\mathbf{a}_i \times \mathbf{b}_i \times \mathbf{d}_i)\ (x-a_i)^2 + (y-b_i)^2 = d_i^2 \qquad (8.1)$$

Figure 8.1 graphically shows an example of the relay problem considering two points (a_1, b_1) and (a_2, b_2). A solution of this problem is a tuple (x, y) such that for all $i = \{1, \ldots, m\}$, $c_i(x, y)$ is true and the goal is to build an inner box $(\mathbf{x}, \mathbf{y})$, in which each point (x, y) is a solution [209].

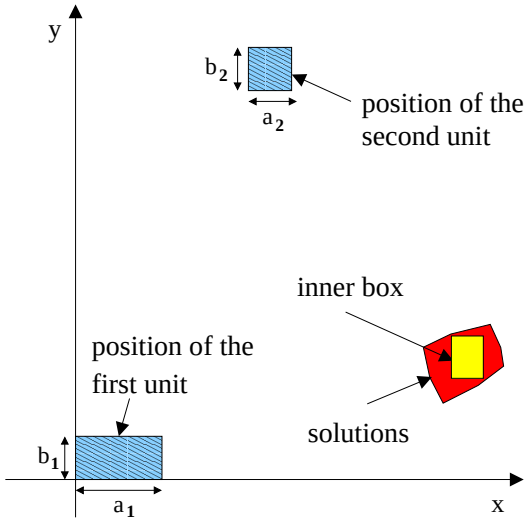


Figure 8.1: The relay positioning problem

As discussed in the previous chapters, classical Interval Analysis and Constraint Programming provide well-known algorithms for handling systems of equations with continuum of solutions [14, 191, 204], but they are not well-adapted for building inner boxes when the system involves existentially quantified parameters (especially when the system is not square w.r.t. the parameters).

Other techniques (as shown in chapters 5 and 7) either based on Modal Intervals [95], or Newton-like existence theorems [72] can detect inner boxes in presence of parameters, but one needs to enforce a whole branch-and-bound process to get an answer.

This chapter proposes an original method for building an inner box around an initial solution of the parameter-free problem. This method starts with a degenerate box (a box reduced to a point, that can be obtained using a least-square method or with the basic approach discussed in Chapter 5, for example) and tries successively to enlarge the dimensions of the box, while proving that the current box remains an inner box.

Domain extension has already been achieved in case of parameter-free inequalities by defining an *univariate* extrema function and computing its left most and right most solutions of a selected variable, using a Newton like method [34].

The new extension algorithm proposed here, works for parametric equations, thus subsuming inequalities and addressing more situations. It essentially extends one variable at a time and the resulting box depends on the order in which variables

are selected. It is based on *arithmetical functions* defined as follows:

Definition 19 (Arithmetical Function) *f is said to be an arithmetical function, if the formal expression $f(x)$ matches the following recursive definition:*

- $f(x) = x_i$, *with* $i = \{1, \ldots, n\}$.
- $f(x) = c$, *where c is a constant in* $\mathbb{R}$.
- $f(x) = \phi(g(x))$, *where g is an arithmetical function, and ϕ is a* elementary *function such as sqr, sqrt, sin, ...*
- $f(x) = g(x) \star h(x)$ *where g and h are arithmetical functions, and $\star$ is a binary operator in* $\{+, -, \times, /\}$.

Informally, the algorithm uses an inner box characterized by a generalized inclusion as $\mathbf{f}(\mathbf{x}) \subseteq [2, -1]$. It is known that, as long as $\mathbf{f}(\mathbf{x}) \subseteq [0, 0]$, $\mathbf{x}$ is an inner box (see sections 3.7.3 and 7.4, for more details). Hence, the idea is to *enlarge* $\mathbf{x}$ as much as possible by considering a right-hand side *enlarged* to $[0, 0]$. The latter enlargement is backward-propagated through the syntactic tree of f down to the leaf representing x (See [15] or Section 4.3.1 for an explanation about *backward propagation*).

8.2 Exploiting the Inner Box Test

As shown in sections 3.7.3 and 7.4.1, by chaining the basic arithmetic operators and functions, one can evaluate any expression with generalized intervals arguments. Moreover, from the Modal Intervals theory (see Section 3.7.3) we have the following proposition:

Proposition 8 *Let $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$ such that each component of v has only one occurrence in $f(x, v)$. Let $\mathbf{x} \in \mathbb{IR}^n$, $\mathbf{v} \in \mathbb{IR}^p$ and $\mathbf{z} = \mathbf{f}(\mathbf{x}, Dual(\mathbf{v}))$. If $\mathbf{z}$ is improper then*

$$(\forall x \in \mathbf{x})(\forall z \in Pro(\mathbf{z}))(\exists v \in \mathbf{v})(z = f(x, v)) \tag{8.2}$$

Up to now, this proposition was mainly used as an *inner box test* (see Section 7.4). As a new result, we will show that Proposition 8 can also be used as a constructive tool for inner boxes.

8.2.1 A General Inner Box Test

Consider a system of m constraints c_i, each constraint being a parametric equation $f_i(x, v) = 0$ with $f_i : \mathbb{R}^n \times \mathbb{R}^p$. Assume that every component v_j appears only once in the whole system. To check if a given box $\mathbf{x}$ is inner, evaluate $\mathbf{f}(\mathbf{x}, Dual(\mathbf{v}))$. The result is a vector $\mathbf{z} \in \mathbb{KR}^m$. If $\mathbf{z} \subseteq 0$ then $\mathbf{x}$ is an inner box.

From Proposition 8, we have

$$(\forall x \in \mathbf{x})(\forall z \in Pro(\mathbf{z}))(\exists v \in \mathbf{v})(z = f(x, v)) \tag{8.3}$$

Since $\mathbf{z} \subseteq 0 \iff 0 \in Pro(\mathbf{z})$ then

$$(\forall x \in \mathbf{x})(\exists v \in \mathbf{v})(f(x, v) = 0) \tag{8.4}$$

8.3 A Generalized Interval Projection

Let us first consider a real-valued arithmetical function $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$. We split variables into $x \in \mathbb{R}$ and $y \in \mathbb{R}^{n-1}$, while $v \in \mathbb{R}^p$ is the vector of parameters. Thus, with no loss of generality, we shall write $f(x, y, v)$.

This section gives a technique to enlarge the domain of a variable that has only one occurrence[1] in the expression of the function[2], with given domains for other variables and parameters. So we assume that x has only one occurrence in f, and fix once for all $\mathbf{y} \in \mathbb{IR}^{n-1}$ and $\mathbf{v} \in \mathbb{IR}^p$.

This technique handles $\mathbf{x}$ (the domain of x) as a variable and tries to find a solution in $\mathbb{KR}$ to some interval relation. We work at the interval level, which must be sharply distinguished from the usual standpoint of interval analysis: Instead of solving an equation of real variable/parameters and using intervals as a way to represent an infinite number of values, we solve an equation of interval variable and look for one interval solution.

To be applied, this technique requires that the variable $\mathbf{x}$ has a domain, i.e., a lower bound and an upper bound, w.r.t. the inclusion order defined in Section 3.1 (Interval Analysis). So there must be intervals $\mathbf{x}_l$ and $\mathbf{x}_u$ such that $\mathbf{x}_l \times \mathbf{y}$ is the initial inner box we want to enlarge, and $\mathbf{x}_u$ is the domain of all possible values

[1] This is a limitation due to the *dependency* problem of interval arithmetic. It can be solved by applying a fixed point algorithm over the multi-occurrence variable, but this is out of scope of this work.

[2] This presentation is done for one constraint. In presence of several constraints, the same operation is performed for each constraint and the intersection of the obtained intervals is returned.

for x. Most of the time, it is easy to provide such an upper bound. Both bounds are proper. We can finally write

$$\mathbf{x}_l \subseteq \mathbf{x} \subseteq \mathbf{x}_u. \tag{8.5}$$

The goal is to find a maximal interval $\mathbf{x} \in \mathbb{KR}$ such that, $\mathbf{x}$ satisfies (8.5) and

$$\mathbf{f}(\mathbf{x}, \mathbf{y}, Dual(\mathbf{v})) \subseteq [0, 0] \tag{8.6}$$

In other words, an interval $\mathbf{x}$ such that both the domain constraint and the inner test are satisfied[3]. Consider now the (slightly) more general problem of finding a maximal $\mathbf{x}$ such that

$$\mathbf{x} \text{ satisfies (8.5)} \quad \textit{and} \quad \mathbf{f}(\mathbf{x}, \mathbf{y}, Dual(\mathbf{v})) \subseteq \mathbf{z} \tag{8.7}$$

with $\mathbf{z} \in \mathbb{KR}$ such that

$$\mathbf{f}(\mathbf{x}_l, \mathbf{y}, Dual(\mathbf{v})) \subseteq \mathbf{z} \subseteq \mathbf{f}(\mathbf{x}_u, \mathbf{y}, Dual(\mathbf{v})). \tag{8.8}$$

Notice that a maximal interval $\mathbf{x}$ satisfying (8.7–8.8) is not necessarily a maximal inner extension of $\mathbf{x}_l$ in $\mathbf{x}_u$. Using Definition 19, we can recursively solve (8.7) by isolating the sub-expression containing x and applying one of the three *elementary* projections detailed below.

8.3.1 Overview

The recursion consists in reducing (8.7–8.8) to a simpler relation

$$\mathbf{x} \text{ satisfies (8.5)} \quad \textit{and} \quad \mathbf{g}(\mathbf{x}, \mathbf{y}, Dual(\mathbf{v})) \subseteq \mathbf{z}', \tag{8.9}$$

where g is a subexpression of f, and $\mathbf{z}'$ satisfies

$$\mathbf{g}(\mathbf{x}_l, \mathbf{y}, Dual(\mathbf{v})) \subseteq \mathbf{z}' \subseteq \mathbf{g}(\mathbf{x}_u, \mathbf{y}, Dual(\mathbf{v})). \tag{8.10}$$

Relation (8.9-8.10) must be a sufficient condition to (8.7-8.8) in the sense that a maximal $\mathbf{x} \in \mathbb{KR}$ satisfying (8.9-8.10) must also be a maximal $\mathbf{x} \in \mathbb{KR}$ satisfying (8.7-8.8). Given f, $\mathbf{x}$, $\mathbf{y}$, $\mathbf{v}$ and $\mathbf{z}$, we detail now how to compute an appropriate $\mathbf{z}'$, dealing with three different cases. These cases are related to the syntactic

[3] If f is linear, some methods already tackle this problem [46, 184, 188, 190].

decomposition of f given by Definition 19. The base case is straightforward. The other cases lie on three concepts: *theoretical projection*, *selection* and *filtering*.

8.3.2 Base case ($\mathbf{x} \subseteq \mathbf{z}$)

By hypothesis, (8.8) holds, i.e., $\mathbf{x}_l \subseteq \mathbf{z} \subseteq \mathbf{x}_u$. Hence, a maximal $\mathbf{x}$ such that $\mathbf{x}$ satisfies (8.5) and ($\mathbf{x} \subseteq \mathbf{z}$) is $\mathbf{z}$ itself.

8.3.3 Basic function ($\phi(g(x, y, v)) \subseteq \mathbf{z}$)

Theoretical projection

For clarity, we replace $\mathbf{g}(\mathbf{x}, \mathbf{y}, \mathit{Dual}(\mathbf{v}))$ by the symbol $\mathbf{g}$. Since every basic function ϕ is piecewise strictly monotonic, hence piecewise invertible, for any $\mathbf{z} \in \mathbb{KR}$, a disjunction of inclusions

$$(\mathbf{g} \subseteq \mathbf{z}_1) \quad \mathit{or} \quad (\mathbf{g} \subseteq \mathbf{z}_2) \quad \mathit{or} \quad \ldots$$

can formally be derived from $\phi(\mathbf{g}) \subseteq \mathbf{z}$, regardless of condition (8.5). For example,

$$\begin{aligned} \exp(\mathbf{g}) \subseteq [1, 2] \quad &\iff \quad \mathbf{g} \subseteq [0, \log(2)] \\ \mathbf{g}^2 \subseteq [4, 0] \quad &\iff \quad \mathbf{g} \subseteq [2, 0] \quad \mathit{or} \quad \mathbf{g} \subseteq [0, -2] \end{aligned}$$

Notice that if $\phi = \mathit{sqr}$, $\mathit{Pro}(\mathbf{z})$ cannot include negative values, so that the square root is always well defined. Indeed, by hypothesis (8.8) holds. If $\mathbf{z}$ is proper, then $\mathbf{z} \subseteq \mathbf{g}(\mathbf{x}_u, \mathbf{y}, \mathit{Dual}(\mathbf{v}))^2$ and $\mathbf{g}(\mathbf{x}_u, \mathbf{y}, \mathit{Dual}(\mathbf{v}))^2 \geq 0$ implies $\mathit{Pro}(\mathbf{z}) \geq 0$. Otherwise, $\mathbf{g}(\mathbf{x}_l, \mathbf{y}, \mathit{Dual}(\mathbf{v}))^2 \subseteq \mathbf{z}$, i.e., $\mathit{Pro}(\mathbf{z}) \subseteq \mathit{Pro}(\mathbf{g}(\mathbf{x}_l, \mathbf{y}, \mathit{Dual}(\mathbf{v}))^2)$ which again implies $\mathit{Pro}(\mathbf{z}) \geq 0$. This symmetry in the domain of *sqrt* and the image of *sqr* is obviously valid for every basic function.

As soon as ϕ is trigonometric, the disjunction includes an infinity of terms (which justifies the *theoretical* qualifier), for example:

$$\cos(\mathbf{g}) \subseteq [0.5, 1] \iff \mathbf{g} \subseteq [-\pi/3, \pi/3] \quad \mathit{or} \quad \ldots$$

All intervals in the (possibly infinite) sequence share both the same proper/improper nature and the same diameter. Furthermore, either their proper projections are all disjoint ($i \neq j \implies \mathit{Pro}(\mathbf{z}_i) \cap \mathit{Pro}(\mathbf{z}_j) = \emptyset$), either they all intersect. They cannot however overlap more than on a bound.

One may wonder if two overlapping intervals $\mathbf{g}_1$ and $\mathbf{g}_2$ can be merged, i.e., if

the condition ($\mathbf{g} \subseteq \mathbf{z}_1$) or ($\mathbf{g} \subseteq \mathbf{z}_2$) can be replaced by $\mathbf{g} \subseteq (\mathbf{z}_1 \vee \mathbf{z}_2)$. This is not allowed since $\mathbf{g} \subseteq (\mathbf{z}_1 \vee \mathbf{z}_2)$ is only a necessary condition[4]. In contrast, $\mathbf{g} \subseteq (\mathbf{z}_1 \wedge \mathbf{z}_2)$ is a sufficient but stronger condition, and maximality is lost (no solution can even be found). Thus, no merging of any kind can be done.

Summing up, solving (8.7) boils down to solving

$$\mathbf{x} \text{ satisfies (8.5)} \quad \textit{and} \quad \mathbf{g}(\mathbf{x}, \mathbf{y}, \mathit{Dual}(\mathbf{v}))) \subseteq \mathbf{z}_j \tag{8.11}$$

for one $\mathbf{z}_j$ in the sequence. We can now avail ourselves of the constraint on the domain of $\mathbf{x}$ to *select* and *filter* a feasible interval in this sequence. *Selection* means that we pick an interval $\mathbf{z}_j$ such that a solution $\mathbf{x}$ of (8.11) exists. *Filtering* means that we find the largest $\mathbf{z}' \subseteq \mathbf{z}_j$ such that (8.10) is satisfied.

Selection

Relation (8.5) allows us to keep only a finite number of z_j in the theoretical projection. Note that by inclusion isotonicity of Kaucher arithmetic, $\mathbf{x}_l \subseteq \mathbf{x}$ implies $\mathbf{g}(\mathbf{x}_l, \mathbf{y}, \mathit{Dual}(\mathbf{v})) \subseteq \mathbf{g}(\mathbf{x}, \mathbf{y}, \mathit{Dual}(\mathbf{v}))$. So it is possible to detect whether $\mathbf{z}_j$ ($j = 1, 2, \ldots$) is feasible or not by checking $\mathbf{g}(\mathbf{x}_l, \mathbf{y}, \mathit{Dual}(\mathbf{v})) \subseteq \mathbf{z}_j$. The number of feasible $\mathbf{z}_j$ resulting from this test is necessarily finite (see Example 8.3.2). We can pick any one of them.

Example 8.3.1 *Consider* $f(x, y, v) = (x + v)^2$, $\mathbf{x}_l = [-1, -1]$, $\mathbf{x}_u = [-2, 3]$, $\mathbf{v} = [-1, 2]$ *and* $\mathbf{z} = [4, 1]$. *Then, we have* $\phi = sqr$, $g(x, y, v) = x + v$ *and*

$$(\mathbf{x} + \mathit{Dual}(\mathbf{v}))^2 \subseteq [4, 1] \iff \begin{cases} \mathbf{x} + \mathit{Dual}(\mathbf{v}) \subseteq [2, 1] \textit{or} \\ \mathbf{x} + \mathit{Dual}(\mathbf{v}) \subseteq [-1, -2] \end{cases}$$

But since $\mathbf{x}_l + \mathit{Dual}(\mathbf{v}) = [1, -2]$, $[2, 1]$ *is not feasible (because* $[1, -2] \not\subseteq [2, 1]$*) whereas* $[-1, -2]$ *is feasible (*$[1, -2] \subseteq [-1, -2]$*).*

For the sake of simplicity, we performed in the last example theoretical projection and selection consecutively, as two separate steps. With trigonometric functions, this is not possible as the number of theoretical projections is infinite. So, we rather use selection as a pre-selecting process. This is illustrated on the next example.

[4] As counter-example, $\mathbf{g} := [-1, 1]$ satisfies $\mathbf{g} \subseteq [0, 2] \vee [-2, 0] = [-2, 2]$ but neither satisfies $\mathbf{g} \subseteq [0, 2]$ nor $\mathbf{g} \subseteq [-2, 0]$.

Example 8.3.2 *Consider* $f(x,y,v) = cos(x+v)$, $\mathbf{x}_l = [6,6]$, $\mathbf{x}_u = [5,9]$, $\mathbf{v} = [-1,1]$ *and. Then, we have* $\phi = cos$ *and* $g(x,y,v) = x+v$. *We first compute*

$$\mathbf{g}_l := \mathbf{x}_l + Dual(\mathbf{v}) = [7,5],$$

It follows that $Pro(\mathbf{g}_l) \subseteq [5,7]$, *which restricts the projection of cosine to two half periods,* $[\pi, 2\pi]$ *and* $[2\pi, 3\pi]$*:*

$$\mathbf{x} + Dual(\mathbf{v}) \subseteq 2\pi + arccos([0.7, 0.8]) = [6.93, 7.08]$$
$$or$$
$$\mathbf{x} + Dual(\mathbf{v}) \subseteq 2\pi - arccos([0.7, 0.8]) = [5.49, 5.64].$$

Filtering

Once $\mathbf{z}_j$ was proven to be feasible, relation (8.5) can be used to make $\mathbf{z}_j$ smaller and fulfill (8.10). Indeed, $\mathbf{x} \subseteq \mathbf{x}_u$ implies $\mathbf{g}(\mathbf{x}, \mathbf{y}, Dual(\mathbf{v})) \subseteq \mathbf{g}(\mathbf{x}_u, \mathbf{y}, Dual(\mathbf{v}))$. Hence we can substitute $\mathbf{z}_j$ by $\mathbf{z}_j \wedge \mathbf{g}(\mathbf{x}_u, \mathbf{y}, Dual(\mathbf{v}))$.

Example 8.3.3 *In Example 8.3.1, we found out that interval* $[-1,-2]$ *was feasible. But as* $\mathbf{x}_u + Dual(\mathbf{v}) = [0,2]$, *we must actually have* $\mathbf{x} + Dual(\mathbf{v}) \subseteq [0,2] \wedge [-1,-2] = [0,-2]$. *This condition is indeed stronger.*

8.3.4 Binary Operator ($g(x,y,v) \star h(y,v) \subseteq \mathbf{z}$)

As $h(y,v)$ is a function in which x does not appear, we simply replace it using an interval $\mathbf{w} := h(\mathbf{y}, Dual(\mathbf{v}))$. Considering first the addition we have:

$$\mathbf{g}(\mathbf{x}, \mathbf{y}, Dual(\mathbf{v})) + \mathbf{w} \subseteq \mathbf{z}$$

and adding $-Dual(\mathbf{w})$ to each side of the latter, we get

$$\mathbf{g}(\mathbf{x}, \mathbf{y}, Dual(\mathbf{v})) \subseteq \mathbf{z} - Dual(\mathbf{w})$$

thanks to the group property of Kaucher arithmetic.

Filtering can apply here to narrow $\mathbf{z} - Dual(\mathbf{w})$. The same idea applies to subtraction and division (by respectively adding and multiplying $\mathbf{z}$ by $Dual(\mathbf{w})$).

Example 8.3.4 *Consider the following expression* $x + y \subseteq z$, *with* $\mathbf{y} = [-1,3]$ *and* $\mathbf{z} = [6,12]$. *Figure 8.2 shows the difference between the classic projection and the generalized projection.*

- *Classic Projection*
  $\mathbf{x} \subseteq \mathbf{z} - \mathbf{y}$
  $\mathbf{x} \subseteq [3, 13]$
- *Generalized Projection*
  $\mathbf{x} \subseteq \mathbf{z} - dual(\mathbf{y})$
  $\mathbf{x} \subseteq [7, 9]$

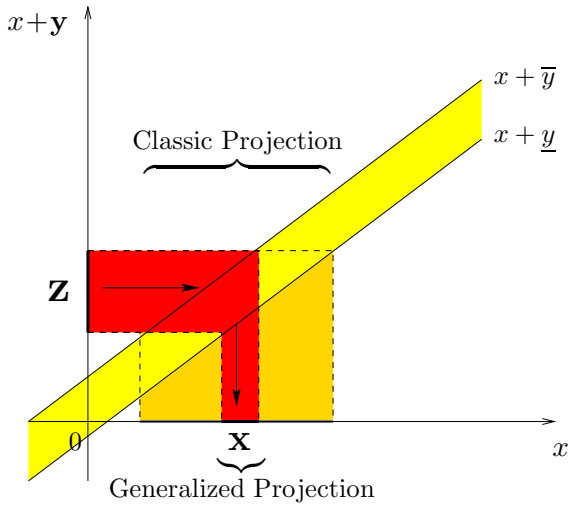


Figure 8.2: An example of the *classic projection* and the *generalized projection*.

Multiplication however requires some precaution. If $0 \notin Pro(\mathbf{w})$, then we can again divide $\mathbf{z}$ by $Dual(\mathbf{w})$. But if $0 \in Pro(\mathbf{w})$, because Kaucher arithmetic does not handle infinite bounds we need to hand-craft a special division. Similar extensions of Kaucher's division are proposed in [71, 166].

Table 8.1 presents the expression for the maximal $\mathbf{g}$ satisfying $\mathbf{g} \times \mathbf{w} \subseteq \mathbf{z}$.

	$\mathbf{z} > \mathbf{0}$	$\mathbf{z} < \mathbf{0}$	$\mathbf{0} \subseteq \mathbf{z}$	$\mathbf{z} \subset \mathbf{0}$
$\mathbf{0} \subseteq \mathbf{w}$	$\emptyset$	$\emptyset$	$[\max\{\underline{\mathbf{z}}/\overline{\mathbf{w}}, \overline{\mathbf{z}}/\underline{\mathbf{w}}\}, \min\{\underline{\mathbf{z}}/\underline{\mathbf{w}}, \overline{\mathbf{z}}/\overline{\mathbf{w}}\}]$	$\emptyset$
$\mathbf{w} \subset \mathbf{0}$	$[-\infty, \underline{\mathbf{z}}/\overline{\mathbf{w}}]$ *or* $[\underline{\mathbf{z}}/\underline{\mathbf{w}}, +\infty]$	$[-\infty, \overline{\mathbf{z}}/\underline{\mathbf{w}}]$ *or* $[\overline{\mathbf{z}}/\overline{\mathbf{w}}, +\infty]$	$[-\infty, +\infty]$	$[-\infty, \min\{\underline{\mathbf{z}}/\overline{\mathbf{w}}, \overline{\mathbf{z}}/\underline{\mathbf{w}}\}]$ *or* $[\max\{\underline{\mathbf{z}}/\underline{\mathbf{w}}, \overline{\mathbf{z}}/\overline{\mathbf{w}}\}, +\infty)]$

Table 8.1: The results of $\mathbf{g}$ for the different cases of $\mathbf{z}$ and $\mathbf{w}$.

Applying filtering on $\mathbf{g}$ immediately removes infinite bounds since $\mathbf{g}$ is necessarily proper. Hence, infinite bounds are not propagated to subsequent computations (which would have led to undefined results). They only are a convenient way to represent arbitrarily large values when enforcing filtering.

Example 8.3.5 *Consider* $f(x, y, v) = x \times v$, $\mathbf{x}_l = [-1, 1]$, $\mathbf{x}_u = [-3, 3]$, $\mathbf{v} = [-1, 2]$ *and* $\mathbf{z} = [-2, 6]$. *Then, thanks to the Table 8.1, we get*

$$\mathbf{x} \subseteq [-\infty, -1] \quad or \quad \mathbf{x} \subseteq [1, +\infty].$$

Both contain $\mathbf{x}_l$*, hence are feasible. Applying* meet *operator with* $\mathbf{x}_u$ *yields*

$$\mathbf{x} \subseteq [-3,-1] \quad or \quad \mathbf{x} \subseteq [1,3]$$

We have seen that the constraint on the domain is crucial in presence of trigonometric functions or multiplication with 0 in operands. In the other cases, by removing domain constraint (i.e., condition (8.5)), it can be easily proven that a maximal $\mathbf{x}$ satisfying

$$\mathbf{f}(\mathbf{x}, \mathbf{y}, Dual(\mathbf{v})) \subseteq [0,0]$$

also satisfies $\mathbf{f}(\mathbf{x}, \mathbf{y}, Dual(\mathbf{v})) = [0,0]$.

It is worth mentioning that functions need not be decomposed formally into sub-expressions: projections are directly performed by an automatic projection algorithm [15], similar to automatic differentiation.

8.4 An Example with the Relay Problem

In this section, the trace of the extension algorithm applied to the relay problem with 4 units is presented. According to the problem in section 8.1, the set of constraints are:

$$c^{(i)}(x,y) : \exists (a_i, b_i, d_i) \in (\mathbf{a}_i \times \mathbf{b}_i \times \mathbf{d}_i)\ (x-a_i)^2 + (y-b_i)^2 = d_i^2$$

and the domains of the parameters are:

$$\begin{array}{lll} \mathbf{a}_1 = [0,2] & \mathbf{b}_1 = [0,1] & \mathbf{d}_1 = [1,8] \\ \mathbf{a}_2 = [4,5] & \mathbf{b}_2 = [9,10] & \mathbf{d}_2 = [1,8] \\ \mathbf{a}_3 = [13,15] & \mathbf{b}_3 = [-11,-10] & \mathbf{d}_3 = [1,14] \\ \mathbf{a}_4 = [16,17] & \mathbf{b}_4 = [5,7] & \mathbf{d}_4 = [1,8] \end{array}$$

A least-square solution obtained by fixing each parameter to the midpoint of its domain is ($\tilde{x} = 9.04286, \tilde{y} = 2.6494$). We first check that this solution can be taken as the starting point of our domain extension. We compute for all i,

$$(\mathbf{x} - Dual(\mathbf{a}_i))^2 + (\mathbf{y} - Dual(\mathbf{b}_i))^2 - Dual(\mathbf{d}_i)^2$$

with $\mathbf{x} = [\tilde{x}, \tilde{x}]$ and $\mathbf{y} = [\tilde{y}, \tilde{y}]$. We get the following image vector :

$$([87.8, -11.7], [78.5, -7.3], [220.8, -20.3], [81.2, -10])$$

As this vector is included in $\mathbf{0}$, then the initial degenerate box $\mathbf{x} \times \mathbf{y}$ is an inner box for the problem. We can now decide that the position (x, y) should not be out of a bounding box $\mathbf{x}_u \times \mathbf{y}_u = [5, 15] \times [0, 20]$. The extension of $\mathbf{x}$ can start.

We detail the projection of $c^{(1)}(x, y)$ over x. Our goal is to find the biggest $\mathbf{x}$ ($\tilde{\mathbf{x}} \subseteq \mathbf{x} \subseteq \mathbf{x}_u$) such that

$$\boxed{(\mathbf{x} - Dual(\mathbf{a}_1))^2 + (\mathbf{y} - Dual(\mathbf{b}_1))^2 - Dual(\mathbf{d}_1)^2 \subseteq 0}$$

Apply Case 3

Compute $\mathbf{w} := (\mathbf{y} - Dual(\mathbf{b}_1))^2 - Dual(\mathbf{d}_1)^2$. We get $\mathbf{w} = (2.6494 - [1, 0])^2 - [8, 1]^2 = [6.02, -61.28]$. Then,

$$(\mathbf{x} - Dual(\mathbf{a}_1))^2 + \mathbf{w} \subseteq 0 \Rightarrow (\mathbf{x} - Dual(\mathbf{a}_1))^2 \subseteq -Dual(\mathbf{w})$$

Finally, $(\mathbf{x} - Dual(\mathbf{a}_1))^2 \subseteq [-6.02, 61.28]$.

We apply domain restriction. We first compute $(\tilde{x} - Dual(\mathbf{a}_1))^2 = [81.77, 49.60]$ and check that $[81.77, 49.60] \subseteq [-6.02, 61.28]$. We also compute $(\mathbf{x}_u - Dual(\mathbf{a}_1))^2 = [25, 169]$ and filter $[-6.02, 61.28]$ to $[25, 61.28]$. Then,

$$\boxed{(\mathbf{x} - Dual(\mathbf{a}_1))^2 \subseteq [25, 61.28]}$$

Apply Case 2

$$(\mathbf{x} - Dual(\mathbf{a}_1)) \subseteq [5, \sqrt{61.28}] = [5, 7.82]$$
$$\vee$$
$$(\mathbf{x} - Dual(\mathbf{a}_1)) \subseteq [-\sqrt{61.28}, -5] = [-7.82, -5]$$

But $(\mathbf{x}_l - Dual(\mathbf{a}_1)) = [9.04, 7.04]$ and $(\mathbf{x}_u - Dual(\mathbf{a}_1)) = [5, 13]$. So, by domain restriction, $[-7.82, -5]$ is discarded, and $[5, 7.82]$ is left intact. Then,

$$\boxed{\mathbf{x} - Dual(\mathbf{a}_1) \subseteq [5, 7.82]}$$

Apply Case 3

$$\mathbf{x} \subseteq [5, 7.82] + \mathbf{a}_1 \iff \boxed{\mathbf{x} \subseteq [5, 9.82]}$$

Apply Case 1

The final answer is $[5, 9.82]$.

We perform a generalized projection to compute consistent extension of $\tilde{x}$ w.r.t the other constraints and get three other intervals:

$$\begin{array}{rcl} \mathbf{x}_{c^{(2)}} \subseteq \sqrt{\mathbf{d}_2^2 - Dual((\tilde{y} - Dual(\mathbf{b}_2))^2)} + \mathbf{a}_2 & = & [5, 9.86] \\ \mathbf{x}_{c^{(3)}} \subseteq \sqrt{\mathbf{d}_3^2 - Dual((\tilde{y} - Dual(\mathbf{b}_3))^2)} + \mathbf{a}_3 & = & [7, 15] \\ \mathbf{x}_{c^{(4)}} \subseteq \sqrt{\mathbf{d}_4^2 - Dual((\tilde{y} - Dual(\mathbf{b}_4))^2)} + \mathbf{a}_4 & = & [8.36, 15] \end{array}$$

The intersection of the four intervals, $[8.36, 9.82]$, is inner w.r.t. the whole system (See Section 7.5).

We can perform now a generalized projection to compute an extension over y of the new box $[8.36, 9.82] \times \tilde{y}$ and we get respectively for each constraint $\mathbf{y}_1 = [0, 2.6494]$, $\mathbf{y}_2 = [2.62, 16]$, $\mathbf{y}_3 = [0, 3.20]$ and $\mathbf{y}_4 = [2.64, 9.35]$. The intersection of these extensions is $[2.6494, 2.6494]$. The final inner box $[8.36, 9.82] \times [2.6494, 2.6494]$ is shown in Figure 8.3(a).

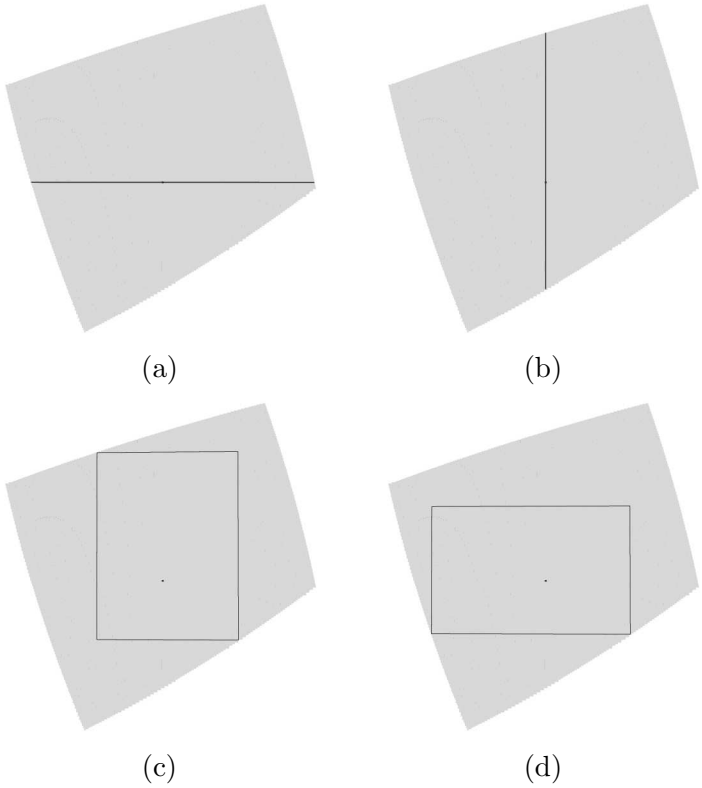


Figure 8.3: (a). Inner box with maximal extension of x and (b) with maximal extension of y. (c) First step of heuristic starting with x. (d) First step of heuristic starting with y.

Let us roll back this extension. If we start domain extension over the variable y at first, and then, over x, we obtain another box, $[9.04286, 9.04286] \times [2.1, 3.42]$ shown in 8.3(b). We observe that the maximal extension obtained for the first variable we project over generally prevents the other variables from being extended. In order to obtain more balanced boxes, we introduce a extension heuristic in two steps.

In the first step, we extend all variables but the last one to the middle point between the initial value and each bound of the maximal extension. The last variable is extended to the maximal interval. For example, using this heuristic x will be extended to $[8.7, 9.43]$ instead of $[8.36, 9.82]$, and then y will be extended to $[2.35, 3.32]$.

In the second step, we perform a maximal extension for all variables (if they can again be extended). Figures 8.3(c) and 8.3(d) show the results obtained with the first step of the heuristic, starting with variable x and variable y (boxes $[8.7, 9.43] \times [2.35, 3.32]$ and $[8.45, 9.48] \times [2.373, 3.039]$), respectively.

Notice that Figure 8.3(c) shows a maximal inner box, while Figure 8.3(d) can be extended again. The second step will extend y to a maximal interval, which is $[2.373, 3.237]$.

8.5 Conclusions

It is clear that for problems having continua of solutions, computing an inner box instead of a solution point is of particular interest, because it gives greater freedom for choosing a solution[5]. Moreover, given a solution point of a problem, it may be interesting to compute a box around this initial solution containing only solutions of the problem. This chapter provided a new method to do it. The key point is the *generalized projection*, a new operator that combines a Constraint Programming concept with theoretical results from Modal Interval Analysis. This projection can be computed in linear time w.r.t. the number of operators and functions involved in the equations. It makes this approach cheap and efficient. Furthermore, universally quantified parameters can also be straightforwardly included.

Some limitations remain: parameters must occur once in the whole system (that means, they must be local to each constraint and appear only once in the constraint), and variables cannot appear more than once in a given equation. This

[5]As noted in [35], in electro-mechanical engineering or civil engineering applications, this approach permits the tolerance of any associated component to be enlarged, and therefore to lower the cost of the components.

last limitation can be overcome by applying a search algorithm to find the maximal $\mathbf{x}$ which verifies $f(x, Dual(\mathbf{v})) \subseteq \mathbf{0}$, but this process is no more a simple evaluation but an iterative algorithm which is more complex and also needs more computing time. Anyway, it may be interesting to study this approach for systems in which only some equations have multi-occurrence variables.

Despite the cited limitations, this is an original and promising approach to handle parametric equations, especially when existentially quantified parameters are involved (problems with uncertainties).

Chapter 9

Conclusions

We studied several strategies for solving systems of equations with uncertainties (specially, distance equations systems) in the framework of Constraint Programming and Interval Analysis. As the uncertainties frequently appear when measurements of physical quantities or experimental data are used as input of a given model, a guaranteed answer to such a model needs to consider the influence of these uncertain values in the final solutions of the problem.

In some situations, it is possible to obtain more information about uncertain values (e.g. an error distribution), but in a general case only a bound for the maximal error is given[1]. In this context, Interval Analysis provides a set of tools for taking into account uncertain data (here represented by intervals), while Constraint Programming contributes with a set of methods and strategies to efficiently solve difficult combinatorial and optimization problems.

The contribution of this thesis is twofold: on the one hand, the introduction of some strategies specifically designed to improve the solving process of systems of distance equations with uncertainty, and on the other hand, the proposition of sufficiently general techniques to be applied to non-linear problems involving uncertain data as parameters.

- **Strategies for solving systems of distance equations**: In Chapter 5 we studied a basic approach for solving systems of distance equations with uncertainty. This approach combines different phases into a specific algorithm which tries to take advantage of the geometry of the solutions

[1]Notice that it is the simplest information that we can have. In a manufacturing process, for example, a good quality control can guarantee that a measure is not more than a given value nor less than another one, but to guarantee an error distribution is quite more difficult to do.

(given for an initial solving process without considering uncertain values) to obtain a sharp approximation of different continua of solutions. As the final phase of this algorithm applies a Branch and Prune strategy to describe a continuum of solutions, we introduce an inner box test to improve the process. Such a test has been studied from different points of view in Chapter 7, introducing the first optimal[2] inner test for distance constraints with existentially quantified parameters.

- **General Strategies for solving problems with uncertainty**: The SOISS algorithm (within the Solver_SOISS) introduced in Chapter 5 is the first approach combining filtering techniques with *conditional* splitting strategies. The advantage of this approach is that it can be used in different situations (not only for problems involving distance constraints) for detecting disjoint continua of solutions without using a later clustering strategy. Maybe the main drawback of this algorithm is the *box representation*[3] used for each generated sub-space. When the continua of solutions cannot be separated in a parallel-to-axis way, it returns a subspace containing more than one continuum. Anyway, if more than one sub-space is generated, the algorithm guarantees that no solution in common exists.

  Still in the framework of continua of solution, Chapter 8 introduced a general approach for building inner boxes, starting with a solution point. The advantage of this approach is that it is not necessary to apply a Branch and Prune process to obtain an inner description of the continuum of solutions. It can be very interesting, for example, for tolerance issues. The limitations of this approach have been also analyzed in the chapter.

As an application to robotics, we presented a novel and efficient algorithm for handling uncertainties in the computation of the direct Kinematics of a 3-2-1 parallel robots (a special class parallel manipulator). We showed that even if an explicit formulation (obtained, for example, with symbolic manipulation of the equations) for the solutions of a problem is available, the use of simple interval evaluation of this expression in order to obtain certified results for a given problem does not produce (in general) thin results (principally due to the

[2]Given a point with uncertainty $\mathbf{a} \in \mathbb{IR}^n$, a distance with uncertainty $\mathbf{r} \in \mathbb{IR}$, and a variable point $\mathbf{x} \in \mathbb{IR}^n$, this test is a necessary and sufficient condition for the following proposition: $(\forall x \in \mathbf{x})(\exists a \in \mathbf{a})(\exists r \in \mathbf{r})(\sum_{i=1}^{n}(x_i - a_i)^2 = r^2)$.

[3]The conditional splitting phase only splits in a parallel-to-axis way. Therefore, a set of interval vectors (boxes) is generated.

wrapping effect and dependency problem of interval arithmetic). For this reason, in Chapter 6 we introduced a four levels algorithm, which combines symbolic manipulation, interval evaluation and constraint propagation. Results show that this combination drastically improves the performance of the solving process, and not only provides a way to compare the influence of different techniques, but also a trade-off between *required precision* and *time limitation*. It can be a very interesting choice for someone who needs fast results (and not so sharp) for some situations, but sharp results for others (paying the precision with computing time).

Another important characteristic of the proposed algorithm is the guarantee of the results. This approach takes into account the uncertain data in each phase of the solving process, providing *certified results* (no solution is lost) for all the levels.

9.1 Future Works

Problems involving continua of solutions are difficult to solve because many of the existential and uniqueness tools do not verify their conditions to be applied. The algorithm SOISS (introduced in Section 5.4) can help to identify disjoint continua of solutions, but the solutions contained inside each computed sub-space need to be analyzed. As evoked in the conclusions of Chapter 5, a combination of *initial solving without uncertainty* and Solver_SOISS algorithm may be of interest to obtain a first approach of the characteristics of the solutions inside each generated sub-space. For example, an important question is: *is the continuum of solutions only a consequence of the uncertain values of the parameters or there exists more than one solution point for an arbitrary value of the parameters?*

Answering such a question means to identify zones in the space in which a unique solution exists, whatever the values of the parameters. A generalization of classic theorems (as for example Kantorovich), to take into account the existentially quantified parameters and guarantee the uniqueness of the solutions (under these conditions) would be a very powerful tool.

On the other hand, if a sharp approximation of the continuum of solutions is needed (for example, to analyze the behavior of the solution space for different values in the parameters of the system), the detection of inner zones is crucial.

The optimal detection of such an inner zone for problems involving existentially quantified parameters is still an open problem. Several works addressing this problem have been proposed in the literature. Most of them are only either

sufficient conditions or necessary ones. In this thesis we presented a test which is both necessary and sufficient condition, but it is specifically designed for distance constraints. The extension of such a test for more general quantified constraints may be very interesting in other areas as, for example, control.

Another open problem is the evaluation of expressions with multi-occurrence of existentially quantified parameters. The evaluation of such a expression (by using generalized interval arithmetic) does not produce an interpretable interval, and thus it is not possible to use this approach for detecting inner boxes. Some recent works (notably those introduced by Goldsztejn [71]) try to overcome this difficulty, but the optimality of such an evaluation is still an open problem.

Bibliography

[1] E. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization*. John Wiley and Sons Ltd., Chichester, England, 1997.

[2] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, 1983.

[3] J. Angeles. Is there a characteristic length of a rigid-body displacement? *Mechanism and Machine Theory*, 41(8):884–896, 2006.

[4] ANSI/IEEE. IEEE standard for binary floating point arithmetic, 1985.

[5] K. Apt and M.G. Wallace. *Constraint Logic Programming using ECLiPSe*. Cambridge University Press, 2006.

[6] J. Armangol, L. Travé-Massuyès, J. Vehí, and M.A. Sainz. Modal interval analysis for error-bounded semiqualitative simulation. In *1r Congrés Català d'Intelligència Artifical (CCIA)*, pages 223–231, 1998.

[7] F. Aurenhammer. Voronoi diagrams-A survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):345–405, 1991.

[8] R. Backofen. Constraint Techniques for Solving the Protein Structure Prediction Problem. *Lecture Notes in Computer Science*, 1520:72–88, 1998.

[9] H. Batnini. *Constraintes Globales et Heuristiques de Recherche pour les CSPs Continus*. PhD thesis, Université de Nice-Sophia Antipolis, 2005.

[10] H. Batnini, C. Michel, and M. Rueher. Mind The Gaps: A New Splitting Strategy for Consistency Techniques. In *Proceedings of Principles and Practice of Constraint Programming (CP 2005), Sitges, Spain*, volume LNCS 3709, pages 77–91. Springer-Verlag, 2005.

[11] H. Batnini and M. Rueher. Semantic decomposition for solving distance constraints. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP 2003)*, pages 964–965, 2003.

[12] H. Batnini and M. Rueher. Quaddist: Filtrage global pour les contraintes de distance. In *10ème Journées Nationales pour la résolution de Problemes NP-Complets (JNPC'04), Angers, France*, pages 59–71, 2004.

[13] F. Benhamou. Interval constraint logic programming. In Andreas Podelski, editor, *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*, pages 1–21. Springer-Verlag, London, UK, 1995.

[14] F. Benhamou and F. Goualard. Universally quantified interval constraints. In *Principles and Practice of Constraint Programming*, pages 67–82, 2000.

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

[16] F. Benhamou, D. McAllester, and P.Van Hentenryck. Clp(intervals) revisited. In *ILPS '94: Proceedings of the 1994 International Symposium on Logic programming*, pages 124–138, Cambridge, MA, USA, 1994. MIT Press.

[17] C. Bessière and M.O. Cordier. Arc-consistency and arc-consistency again. In Manfred Meyer, editor, *Proceedings ECAI'94 Workshop on Constraint Processing*, Amsterdam, 1994.

[18] C. Bessière, P. Meseguer, E.C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. In *Principles and Practice of Constraint Programming*, pages 88–102, 1999.

[19] C. Bessière and J.C. Régin. MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings of Principles and Practice of Constraint Programming*, pages 61–75, 1996.

[20] C. Bessière and J.C. Régin. Arc-consistency for general constraint networks: Preliminary results. In *Proceedings of IJCAI*, pages 398–404, 1997.

[21] L.M. Blumenthal. *Theory and Applications of Distance Geometry*. Chelsea, New York, 1970.

[22] B. Bouchon-Meunier and V. Kreinovich. From interval computations to modal mathematics: applications and computational complexity. *SIGSAM Bull.*, 32(2):7–11, 1998.

[23] H. Brönnimann, G. Melquiond, and S. Pion. A proposal to add interval arithmetic to the c++ standard library. Research Report 5646, INRIA, July 2005.

[24] C. Brown. Quantifier Elimination by Partial Cylindrical Algebraic Decomposition, `http://www.cs.usna.edu/~qepcad/B/QEPCAD.html`.

[25] O. Caprani, K. Madsen, and H.B. Nielsen. Introduction to interval analysis. lecture notes 2002, `http://www2.imm.dtu.dk/courses/02611/`.

[26] Y. Caseau and F. Laburthe. CLAIRE: Combining objects and rules for problem solving. In *Proceedings of the JICSLP'96 workshop on multi-paradigm logic programming*, pages 105–114, 1996.

[27] A. Cayley. A theorem in the geometry of position. *Cambridge Mathematical Journal*, 2:267–271, 1841.

[28] M. Ceccarelli, M.E. Toti, and E. Ottaviano. CATRASYS (Cassino Tracking System): A new measuring system for workspace evaluation of robots. In *8th International Workshop on Robotics in Alpe-Adria-Danube Region RAAD'99, Munich*, pages 19–24, 1999.

[29] Celesco. *User's Manual for Wire Transducers mod. PT101.* Celesco Inc., 1994.

[30] G. Chabert, G. Trombettoni, and B. Neveu. Box-set consistency for interval-based constraint problems. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1439–1443, New York, NY, USA, 2005. ACM Press.

[31] Y. Chen. Improving Han and Lee's path consistency algorithm. In *In Proceedings of the 3rd IEEE International Conference on Tools for AI*, pages 346–350, 1991.

[32] H. Chun. Constraint programming in java with jsolver. In *Proceedings of the First International Conference and Exhibition on the Practical Application of Constraint Technologies and Logic Programming*, London, 1999.

[33] J.G. Cleary. Logical arithmetic. *Future Computing Systems*, 2(2):125–149, 1987.

[34] H. Collavizza, F. Delobel, and M. Rueher. Comparing partial consistencies. *Reliable Computing*, 1:1–16, 1999.

[35] H. Collavizza, F. Delobel, and M. Rueher. Extending consistent domains of numeric CSP. In *Proceedings of IJCAI*, pages 406–413, 1999.

[36] George Collins. Quantifier elimination for the elementary theory of real closed fields by cynlindrical algebraic decomposition. In *Automata Theory and Formal Languages, Berlin, Germany*, volume LNCS 33, pages 264–274. Springer-Verlag, 1975.

[37] George Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *J. Symb. Comput.*, 12(3):299–328, 1991.

[38] A. Colmerauer. Solving equations and inequations on finite and infinite trees. In *Proceedings of the Conference on Fifth Generation Computer Systems*, pages 85–99, Tokyo, 1984.

[39] A. Colmerauer. Prolog in 10 figures. *Commun. ACM*, 28(12):1296–1310, 1985.

[40] A. Colmerauer, H. Kanoui, R. Pasero, and P. Roussel. Un système de communication en français. Technical report, Groupe Intelligence Artificielle, Faculté de Sciences de Luminy, Université Aix-Marseille II, France, 1972.

[41] A. Connell and R. Corless. An experimental interval arithmetic package in maple. *Interval Computations*, (2):120–134, 1993.

[42] M.C. Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41(1):89–95, 1989.

[43] G. Corliss. INTPAK for interval arithmetic in maple : Introduction and applications. *Submitted to Journal of Symbolic Computation*, 1994.

[44] G.M. Crippen and T.F. Havel. Distance geometry and molecular conformation. *Quarterly Review of Biology*, 64(4), 1989.

[45] A. Cuyt, B. Verdonk, S. Becuwe, and P. Kuterna. A remarkable example of catastrophic cancellation unraveled. *Computing*, 66(3):309–320, 2001.

[46] S. Markov d E. Popova and Ch. Ulrich. On the Solution of Linear Algebraic Equations Involving Interval Coefficients. *Iterative Methods in Linear Algebra, II*, 3:216–225, 1996.

[47] G.B. Dantzig. Programming in a linear structure. *Econometrica*, 17:73–74, 1949.

[48] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 32(3):281–331, 1987.

[49] R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proceedings of IJCAI*, volume 1, pages 412–417. Morgan Kaufmann, 1997.

[50] R. Debruyne and C. Bessière. Domain filtering consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.

[51] R. Dechter. Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, 1990.

[52] O. Didrit. *Analyse par intervalles pour l'Automatique; Résolution globale et garantie des problèmes non linéaires en robotique et en commande robuste*. PhD thesis, Université Paris sud. UFR Scientifique d'Orsay, 1997.

[53] A. Dolzmann. *Reelle Quantorenelimination durch parametrisches Zählen von Nullstellen*. PhD thesis, FMI, Universität Passau, D-9403 Passau, Germany, November 1994.

[54] A. Dolzmann and T. Sturm. Redlog user manual. Technical Report MIP-9616, FMI, Universität Passau, D-94030 Passau, Germany, October 1996.

[55] A. Dolzmann, T. Sturm, and V. Weispfenning. Real quantifier elimination in practice. Technical Report MIP9720, FMI, Universität Passau, D-94030 Passau, Germany, December 1997.

[56] Kaucher E. über Eigenschaften und Anwendungsmöglichkeiten der erweiterten Intervallrechnung und des hyperbolischen Fastkörpers über $\mathbb{IR}$. *Computing*, Suppl. 1:81–94, 1977.

[57] Kaucher E. Interval Analysis in the Extended Interval Space $\mathbb{IR}$. *Computing*, Suppl. 2:33–49, 1980.

[58] A.J. Fernández and P.M. Hill. A comparative study of eight constraint programming languages over the boolean and finite domains. *Constraints*, 5(3):275–301, 2000.

[59] R.E. Fikes. REF-ARF: A System for Solving Problems Stated as Procedures. *Artificial Intelligence*, 1:27–120, 1970.

[60] E.C. Freuder. Synthesizing constraint expressions. *Commun. ACM*, 21(11):958–966, 1978.

[61] A. Frommer, B. Lang, and M. Schnurr. A comparison of the Moore and Miranda existence tests. *Computing*, 72(3-4):349–354, 2004.

[62] T. Frühwirth. Constraint handling rules. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, volume 910 of *Lecture Notes in Computer Science*, page 90. Springer-Verlag, 1995.

[63] T. Frühwirth, A. Herold, V. Küchenhoff, T. Le Provost, P. Lim, E. Monfroy, and M. Wallace. Constraint logic programming - an informal introduction. Technical report, European Computer-Industry Research Center, 1993.

[64] E. Gardeñes, H. Mielgo, and A. Trepat. Modal Intervals: Reason and Ground Semantics. In *Proceedings of the International Symposium on Interval Mathemantics 1985*, pages 27–35, London, UK, 1986. Springer-Verlag.

[65] E. Gardeñes, M.Á. Sainz, L. Jorba, R. Calm, R. Estela, H. Mielgo, and A. Trepat. Modal intervals. *Reliable Computing*, 1(2):77–111, 2001.

[66] E. Gardeñes and A. Trepat. Fundamentals of SIGLA, an interval computing system over the completed set of intervals. *Computing*, 24(2–3):161–179, 1980.

[67] J.G. Gaschnig. *Performance measurement and analysis of certain search algorithms*. PhD thesis, Carnegie Mellon University, Pittsburgh, 1979.

[68] Z.J. Geng and L.S. Haynes. A 3-2-1 kinematic configuration of a stewart platform and its application to six degree of freedom pose measurements. *Robotics and Computer-Integrated Manufacturing*, 11(1):23–34, 1994.

[69] F. Glover. Tabu search. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[70] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, 1991.

[71] A. Goldsztejn. *Définition et Applications des Extensions des Fonctions Réelles aux Intervalles Généralisés*. PhD thesis, Université de Nice-Sophia Antipolis, 2005.

[72] A. Goldsztejn. A Branch and Prune Algorithm for the Approximation of Non-Linear AE-solution Sets. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1650–1654, New York, NY, USA, 2006. ACM Press.

[73] A. Goldsztejn and G. Chabert. On the approximation of linear AE-solution sets. In *Proc. of the 12th GAMM - IMACS International Symposion on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN-2006), Duisburg, Germany*, 2006.

[74] A. Goldsztejn and L. Jaulin. Inner and outer approximations of existentially quantified equality constraints. In *Proceedings of Principles and Practice of Constraint Programming (CP 2006), Nantes, France*, volume LNCS 4204, pages 198–212. Springer-Verlag, 2006.

[75] C. Gosselin, J. Sefrioui, and M.J. Richard. Solution polynomiale au problème de la cinématique directe des manipulateurs parallèles plans à 3 degrés de liberté. *Mechanism and Machine Theory*, 27(2):107–119, 1992.

[76] C. Grandón, G. Chabert, and B. Neveu. Generalized Interval Projection: A New Technique for Consistent Domain Extension. In *Twentieth International Joint Conferences on Artificial Intelligence (IJCAI-07)*, pages 94–99, 2007.

[77] C. Grandón, D. Daney, and Y. Papegay. Combining CP and interval methods for solving the direct kinematic of a parallel robot under uncertainties. In *Proc. of the Workshop IntCP of Twelfth International Conference on Principles and Practice of Constraint Programming (CP-2006), Nantes, France*, 2006.

[78] C. Grandón, D. Daney, C. Tavolieri, E. Ottaviano, and M. Ceccarelli. A Combination of Symbolic and Numerical Solvers for Handling Uncertainties for a Class of Parallel Robots. Accepted for publication to the 12th World

Congress in Mechanism and Machine Science (IFToMM-2007), Besançon, France, 2007.

[79] C. Grandón and A. Goldsztejn. Inner Aproximation of Distance Constraints with Existentially Quantified Parameters. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1660–1661, New York, NY, USA, 2006. ACM Press.

[80] C. Grandón and A. Goldsztejn. Quantifier Elimination versus Generalized Interval Evaluation: A comparison on a special class of quantified constraints. In *Proc. of the 11th Information Processing and Management of Uncertainty International Conference, IPMU 2006*, pages 786–793, Paris, France, 2006. Editions EDK.

[81] C. Grandón and B. Neveu. Using Constraint Programming for Solving Distance CSP with Uncertainty. In *Principles and Practice of Constraint Programming - CP 2005: 11th International Conference, CP 2005, Sitges, Spain*, volume LNCS 3709. Springer-Verlag, 2005.

[82] C. Grandón and B. Neveu. A Specific Quantifier Elimination for Inner Box Test in Distance Constraints with Uncertainties. Research Report 5883, INRIA, April 2006.

[83] L. Granvilliers. RealPaver user's manual, August 2004.

[84] SIGLA/X group. Análisis intervalar clásico (white paper in spanish) `http://ima.udg.es/~sainz/cuadernosint.html`.

[85] SIGLA/X group. Modal intervals (basic tutorial). *Applications of Interval Analysis to Systems and Control (Proceedings of MISC'99)*, pages 157–227, 1999.

[86] C-C Han and C-H Lee. Comments on mohr and henderson's path consistency algorithm. *Artificial Intelligence*, 36(1):125–130, 1988.

[87] E. Hansen. *Global optimization using interval analysis.* Marcel Deckler, 1992.

[88] E. Hansen and R. Greenberg. An interval newton method. *Applied Mathematics and Computation*, 12:89–98, 1983.

[89] E. Hansen and S. Sengupta. Bounding solutions of systems of equations using interval analysis. *BIT Numerical Mathematics*, 21(2):203–211, 1981.

[90] E. Hansen and G.W. Walster. *Global Optimization Using Interval Analysis. Second Edition, Revised and Expanded.* Marcel Deckler, 2003.

[91] R. Haralick and G. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[92] B. Hayes. A Lucid Interval. *American Scientist*, 91(6):484–488, 2003.

[93] P.Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge, MA, USA, 1989.

[94] P.Van Hentenryck. A gentle introduction to NUMERICA. *Artificial Intelligence*, 103(1-2):209–235, 1998.

[95] P. Herrero, M.A. Sainz, J. Vehí, and L. Jaulin. Quantified set inversion algorithm with applications to control. In *Proceedings of IMCP'04 (Interval Mathematics and Constrained Propagation methods), Novosibirsk, 2004*, volume 5 of *Reliable Computing*, 2004.

[96] P. Herrero, M.A. Sainz, J. Vehí, and L. Jaulin. Quantified set inversion with applications to control. In *IEEE International Symposium on Computer Aided Control Systems Design*, 2004.

[97] T. Hickey, Q. Ju, and M.H. Van Emden. Interval arithmetic: From principles to implementation. *J. ACM*, 48(5):1038–1068, 2001.

[98] Hoon Hong, Richard Liska, and Stanly Steinberg. Testing stability by quantifier elimination. *Journal of Symbolic Computation*, 24(2):161–187, 1997.

[99] K.H. Hunt and E.J.F. Primrose. Assembly configurations of some in-parallel actuated manipulators. *Mechanism and Machine Theory*, 28(1):31–42, 1993.

[100] E. Hyvönen. Constraint reasoning based on interval arithmetic. In *Proceedings of IJCAI*, pages 1193–1198, 1989.

[101] E. Hyvönen and S. De Pascale. Interval computations on the spreadsheet. In *Applications of Interval Computations, Kluwer*, pages 169–209, 1996.

[102] E. Hyvönen and S. De Pascale. A new basis for spreadsheet computing: Interval solver for microsoft excel. In *Proceedings of the sixteenth national conference on Artificial Intelligence AAAI'99/IAAI'99*, pages 799–806, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[103] IEEE. Draft standard for floating-point arithmetic p754, august 2006, `http://754r.ucbtest.org/drafts/754r.pdf`.

[104] ILOG. Ilog Solver, reference manual, August 2000.

[105] ILOG. Ilog Solver, user's manual, August 2000.

[106] J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[107] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis with Examples in Parameter and State Estimation, Robust Control and Robotics*. Springer-Verlag, 2001.

[108] L. Jaulin, S. Ratschan, and L. Hardouin. Set computation for nonlinear control. *Reliable Computing*, 10(1):1–26, 2004.

[109] W.M. Kahan. A more complete interval arithmetic. Technical report, University of Toronto, 1968.

[110] V. Kantorovich. Functional Analysis ans Applied Mathematics, translated by C. D. Benster, National Bureau of Standards, Report 1509, 1952.

[111] E. Kaucher. *Über metrische und algebraische Eigenschaften einiger beim numerischen Rechnen auftretender Räume*. PhD thesis, Universität Karlsruhe, Germany, 1973.

[112] R.B. Kearfott, M. Dawande, K. Du, and C. Hu. Intlib: A portable fortran-77 elementary function library. *Interval Computations*, 3:96–105, 1992.

[113] R.B. Kearfott, M. Dawande, K. Du, and C. Hu. Algorithm 737: Intlib—a portable fortran 77 interval standard-function library. *ACM Trans. Math. Softw.*, 20(4):447–459, 1994.

[114] R.B. Kearfott, J. Dian, and A. Neumaier. Existence verification for singular zeros of complex nonlinear systems. *SIAM J. Numer. Anal.*, 38(2):360–379, 2000.

[115] R.B. Kearfott and M. Novoa. Algorithm 681: INTBIS, a portable interval Newton/bisection package. *ACM Trans. Math. Softw.*, 16(2):152–157, 1990.

[116] J. Keiper. Interval arithmetic in mathematica. *Interval Computations*, (3):76–87, 1993.

[117] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[118] O. Knuppel. PROFIL/BIAS v2.0, february 1999, (library and user manual) `http://www.ti3.tu-harburg.de/knueppel/profil/index_e.html`.

[119] O. Knuppel. PROFIL/BIAS – a fast interval library. *Computing*, 53(3-4):277–287, 1994.

[120] W. Krämer and I. Geulig. Interval calculus in maple - the extension intpakx to the package intpak of the share-library. Preprint 2001/2, Universität Wuppertal, 2001.

[121] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4:187–201, 1969.

[122] R. Krawczyk and A. Neumaier. Interval slopes for rational functions and associated centered forms. *SIAM Journal on Numerical Analysis*, 22(3):604–616, 1985.

[123] R. Krawczyk and A. Neumaier. An improved interval Newton operator. *Journal of mathematical analysis and applications*, 118(1):194–207, 1986.

[124] V. Kreinovich, D.J. Berleant, R. Joan-Arinyo, and M. Koshelev. Interval computations, `http://www.cs.utep.edu/interval-comp/`.

[125] V. Kreinovich, A.V. Lakeyev, and S.I. Noskov. Optimal solution of interval linear systems is intractable (NP-hard). *Interval Computations*, (1):6–14, 1993.

[126] V. Kreinovich, V.M. Nesterov, and N.A. Zheludeva. Interval methods that are guaranteed to underestimate (and the resulting new justification of Kaucher arithmetic). *Reliable Computing*, 2(2):119–124, 1996.

[127] L. Krippahl and P. Barahona. Applying Constraint Programming to Protein Structure Determination. In *Proc. of 5th International Conference on*

Principles and Practice of Constraint Programming (CP'99), volume 1713 of *Lecture Notes in Computer Science*, pages 289–302, 1999.

[128] V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, 13(1):32–44, 1992.

[129] J.-L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10(1):29–127, 1978.

[130] Y. Lebbah and O. Lhomme. Accelerating filtering techniques for numeric CSPs. *Artificial Intelligence*, 139(1):109–132, 2002.

[131] Y. Lebbah, M. Rueher, and C. Michel. A global filtering algorithm for handling systems of quadratic equations and inequations. volume 2470 of *Lecture Notes in Computer Science*, pages 109–123, 2002.

[132] M. Lerch, G. Tischler, J.W. von Gudenberg, W. Hofschuster, and W. Krämer. FILIB++, a fast interval library supporting containment computations. *ACM Transactions on Mathematical Software*, 32(2):299–324, 2006.

[133] O. Lhomme. Consistency techniques for numeric csps. In *Proceedings of IJCAI*, pages 232–238, 1993.

[134] E. Loh and G.W. Walster. Rump's example revisited. *Reliable Computing*, 8(3):245–248, 2002.

[135] I.J. Lustig and J.-F. Puget. Program does not equal program: Constraint programming and its relationship to mathematical programming. *Interfaces*, 31(6):29–53, 2001.

[136] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[137] D.E. Manolakis. Efficient solution and performance analysis of 3-d position estimation by trilateration. *IEEE Transactions on Aerospace and Electronic Systems*, 32(4):1239–1248, 1996.

[138] S.M. Markov. On directed interval arithmetic and its applications. *J. UCS*, 1(7):514–526, 1995.

[139] K. Menger. New foundation for euclidean geometry. *American Journal of Mathematics*, (53):721–745, 1931.

[140] J.-P. Merlet. Assembly modes and direct kinematics of parallel manipulators. In *ISRAM*, volume 3, pages 43–48, Burnaby, 1990. ASME Press Series.

[141] J.-P. Merlet. Analyse par intervalles et applications (cours in french), `http://www-sop.inria.fr/coprin/logiciels/ALIAS/Examples/COURS/`, 2003.

[142] J.-P. Merlet. ALIAS: An Algorithms Library of Interval Analysis for equation Systems, September 2004.

[143] J.-P. Merlet. Solving the forward kinematics of a Gough-type parallel manipulator with interval analysis. *I. J. Robotic Res.*, 23(3):221–235, 2004.

[144] J.-P. Merlet. *Parallel robots, Second edition.* Springer-Verlag New York Inc, 2006.

[145] J.-P. Merlet and D. Daney. A formal-numerical approach to determine the presence of singularity within the workspace of a parallel robot. *Computational Kinematics*, pages 167–176, 2001.

[146] Z. Michalewicz. *Genetics Algorithms + Data Structures = Evolution Programs.* WNT, Warsaw, 1996.

[147] D. Michelucci and S. Foufou. Using cayley-menger determinants for geometric constraint solving. In *Proceedings de ACM Symposium on Solid Modeling and Application, Genova, Italy*, pages 285–290, 2004.

[148] C. Miranda. Un' osservazione su un teorema di Brouwer. *Bol. Un. Mat. Ital.*, 3:5–7, 1940.

[149] R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[150] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[151] R.E. Moore. Automatic error analysis in digital computation. Technical report, LMSD-48421, Lockheed Missiles and Space Division, Sunnyvale, CA, 1959.

[152] R.E. Moore. *Interval arithmetic and automatic error analysis in digital computing*. PhD thesis, Stanford University, 1962.

[153] R.E. Moore. *Interval Analysis*. Prentice Hall, 1966.

[154] R.E. Moore. A test for existence of solutions to nonlinear systems. *SIAM J. Numer. Anal.*, 14(4):611–615, 1977.

[155] R.E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.

[156] H. Muñoz and R.B. Kearfott. Slope intervals, generalized gradients, semigradients, slant derivatives, and csets. *Reliable Computing*, 10(3):163–193, 2004.

[157] B.A. Nadel. Tree search and arc consistency in constraint satisfaction algorithms. pages 287–342, 1988.

[158] P. Nanua and K.J. Waldron. Direct kinematic solution of a stewart platform. *IEEE Trans. on Robotics and Automation*, 6(4):438–444, 1991.

[159] A. Neumaier. *Interval Methods for Systems of Equations*, volume 37. Encyclopedia of Mathematics and its Applications, 1990.

[160] A. Neumaier. Taylor forms-use and limits. *Reliable Computing*, 9(1):43–79, 2003.

[161] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. In *Acta Numerica 2004 (A. Iserles, ed.), Cambridge University Press*, pages 271–369, 2004.

[162] E. Ottaviano, M. Ceccarelli, M. Toti, and C. Avila-Carrasco. CATRASYS (Cassino Tracking System): A wire system for experimental evaluation of robot workspace. *Journal of Robotics and Mechatronics*, 14(1):78–87, 2002.

[163] Ch.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, Englewood Cliffs, NJ, 1982.

[164] E. Popova. Generalizing the parametric fixed-point iteration. In *Proceedings in Applied Mathematics & Mechanics (PAMM)*, volume 4, pages 680–681, 2004.

[165] E. Popova. Improved solution enclosures for over- and underdetermined interval linear systems. In *Proceedings of LSSC 2005*, volume LNCS 3743, pages 305–312. Springer-Verlag, 2006.

[166] E.D. Popova. Extended Interval Arithmetic in IEEE Floating-Point Environment. *Interval Computations*, (4):100–129, 1994.

[167] J.M. Porta, L. Ros, F. Thomas, and C. Torras. A Branch-and-Prune Algorithm for Solving Systems of Distance Constraints. In *Proc. of the 2003 IEEE International Conference on Robotics and Automation (ICRA'03), Taipei, Taiwan*, May 2003.

[168] P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In R. Dechter, editor, *Proceedings of Principles and Practice of Constraint Programming*, pages 353–368. Springer Verlag, 2000.

[169] J.F. Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of AAAI'98/IAAI'98*, pages 359–366, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[170] D. Ratz. Box-splitting strategies for the interval Gauss–Seidel step in a global optimization method. *Computing*, 53:337–354, 1994.

[171] D. Ratz. Inclusion isotone extended interval arithmetic. a toolbox update. Technical report, Institut für Angewandte Mathematik, Universität Karlsruhe, Germany, 1996.

[172] D. Ratz. An optimized interval slope arithmetic and its application. Technical report, Institut für Angewandte Mathematik, Universität Karlsruhe, Germany, 1996.

[173] J-C Régin. Ac-*: A configurable, generic and adaptive arc consistency algorithm. In *Proceedings of Principles and Practice of Constraint Programming (CP 2005), Sitges, Spain*, volume LNCS 3709, pages 505–519. Springer-Verlag, 2005.

[174] J.C. Régin. A filtering algorithm for constraints of difference in csps. In *Proceedings of the twelfth national conference on Artificial Intelligence (AAAI '94)*, pages 362–367, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[175] C.M. Rekkas, C.C. Lefas, and N.J. Krikelis. Improving the accuracy of aircraft absolute altitude estimation using dme measurements. *International journal of systems science*, 21(7):1381–1392, 1990.

[176] N. Revol. Introduction à l'arithmétique par intervalles. Rapport de Recherche 4297, INRIA Rhône-Alpes, 2001.

[177] E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill Higher Education, 1990.

[178] J. Rohn and V. Kreinovich. Computing exact componentwise bounds on solutions of linear systems with interval data is NP-hard. *SIAM Journal on Matrix Analysis and Applications (SIMAX)*, 16:415–420, 1995.

[179] J.G. Rokne and P. Bao. Interval Taylor forms. *Computing*, 39(3):247–259, 1987.

[180] Francesca Rossi, Charles Petrie, and Vasant Dhar. On the equivalence of constraint satisfaction problems. In Luigia Carlucci Aiello, editor, *ECAI'90: Proceedings of the 9th European Conference on Artificial Intelligence*, pages 550–556, Stockholm, 1990. Pitman.

[181] S.M. Rump. Algorithms for verified inclusions – theory and practice. *Reliability in computing*, 19:109–126, 1988.

[182] S.M. Rump. A note on epsilon-inflation. *Reliable Computing*, 4:1–5, 1998.

[183] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, editor, *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, PPCP'94, Washington, USA*, pages 10–20, 1994.

[184] M.Á. Sainz, E. Gardeñes, and L. Jorba. Formal Solution to Systems of Interval Linear or Non-Linear Equations. *Reliable Computing*, 8(3):189–211, 2002.

[185] D. Sam-Haroud and B. Faltings. Consistency techniques for continuous constraints. *Constraints*, 1(1/2):85–118, 1996.

[186] J. Sam-Haroud. *Constraint Consistency Techniques for Continuous Domains*. Phd. thesis no. 1423, École Polytechnique Fédérale de Lausanne, Lausanne (Switzerland), 1995.

[187] Jacob Schwartz and Micha Sharir. On the 'piano movers' problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. Technical Report 39, Department of Computer Science, Courant Institute of Mathematical Sciences, 1981.

[188] S.P. Shary. Algebraic Approach to the Interval Linear Static Systems. *Reliable Computing*, 3(1):3–33, 1996.

[189] S.P. Shary. Interval Gauss-Seidel method for generalized solution sets to interval linear systems. *Reliable Computing*, 7(2):141–155, 2001.

[190] S.P. Shary. A new technique in systems analysis under interval uncertainty and ambiguity. *Reliable Computing*, 8(5):321–418, 2002.

[191] M.C. Silaghi, D. Sam-Haroud, and B. Faltings. Search techniques for non-linear constraint satisfaction problems with inequalities. In *Proc. of 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, volume 2056 of *Lecture Notes in Computer Science*, pages 183–193, 2001.

[192] G. Smolka. The oz programming model. In Jan van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 324–343. Springer-Verlag, Berlin, 1995.

[193] V. Stahl. *Interval methods for bounding the range of polynomials and solving systems of nonlinear equations*. PhD thesis, Research Institute for Symbolic Computation, University Linz, Germany, 1995.

[194] R. Steiner, H. Kaindl, and G. Kainz. Backjumping in state-space search. In *Proceedings European Conference on Artificial Intelligence, ECAI-96*, pages 395–399, 1996.

[195] T. Sunaga. Theory of interval algebra and its applications to numerical analysis. *RAAG Memoirs*, 2:29–46, 1958.

[196] R.A. Tapia. The kantorovich theorem for Newton's method. *American Mathematic Monthly*, 78(1):389–392, 1971.

[197] F. Thomas. Solving geometric constraints by iterative projections and backprojections. In *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA'04)*, volume 2, pages 1789–1794, 2004.

[198] F. Thomas, E. Ottaviano, L. Ros, and M. Ceccarelli. Performance analysis of a 3-2-1 pose estimation device. *IEEE Transactions on Robotics*, 21(3):288–297, 2005.

[199] J. Touati. Traitement des incertitudes dans les contraintes de distance. Rapport de stage scientifique, ENPC, 2002.

[200] M.R.C. van Dongen. Ac-3d an efficient arc-consistency algorithm with a low space-complexity. In *CP '02: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, pages 755–760, London, UK, 2002. Springer-Verlag.

[201] W.J. van Hoeve. The alldifferent constraint: A survey, 2001.

[202] R. Vertechy and V. Parenti-Castelli. An accurate and fast algorithm for the determination of the rigid body pose by three point position data. In *International Worshop on Computational Kinematics*, 2005.

[203] X-H. Vu, D. Sam-Haroud, and B. Faltings. Clustering for disconnected solution sets of numerical csps. In *Recent Advances in Constraints: Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2003, Budapest*, volume LNAI 3010. Springer-Verlag, 2004.

[204] X-H Vu, D. Sam-Haroud, and M.C. Silaghi. Approximation techniques for non-linear problems with continuum of solutions. In *Proceedings of The 5th International Symposium on Abstraction, Reformulation and Approximation*, volume LNAI 2371, Canada, August 2002.

[205] X-H. Vu and D. Sam-Haroud M.C. Silaghi. Numerical constraint satisfaction problems with non-isolated solutions. In *1st International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS'2002)*, France, October 2002.

[206] X.-H. Vu, M.C. Silaghi, D. Sam-Haroud, and B. Faltings. Branch-and-Prune Search Strategies for Numerical Constraint Solving. Submited to ACM Transactions on Computational Logic, 2006.

[207] G.W. Walster and E.R. Hansen. Interval algebra, composite functions and dependence in compilers, white paper available at `http://www.mscs.mu.edu/~globsol/Papers/composite.ps`, 1998.

[208] D.L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1972.

[209] A.C. Ward, T. Lozano-Perez, and W.P. Seering. Extending the constraint propagation of intervals. In *Proceedings of IJCAI*, pages 1453–1458, 1989.

[210] M. Warmus. Calculus of approximations. *Bulletin de Sciences, L'Academie Polonaise de Sciences*, 4(5):253–259, 1956.

[211] K. Yamamura, H. Kawata, and A. Tokue. Interval solution of nonlinear equations using linear programming. *BIT*, 38(1):186–199, 1998.

[212] R.C. Young. The algebra of many-valued quantities. *Mathematische Annalen*, 104:260–290, 1931.

[213] Z. Yuanlin. *Consistency Techniques in Constraint Networks*. PhD thesis, National University of Singapore, 2003.