

# Efficient handling of universally quantified inequalities

Alexandre Goldsztejn · Claude Michel · Michel Rueher

Published online: 20 July 2008  
© Springer Science + Business Media, LLC 2008

**Abstract** This paper introduces a new framework for solving quantified constraint satisfaction problems (QCSP) defined by universally quantified inequalities on continuous domains. This class of QCSPs has numerous applications in engineering and technology. We introduce a generic branch and prune algorithm to tackle these continuous CSPs with parametric constraints, where the pruning and the solution identification processes are dedicated to universally quantified inequalities. Special rules are proposed to handle the parameter domains of the constraints. The originality of our framework lies in the fact that it solves the QCSP as a non-quantified CSP where the quantifiers are handled locally, at the level of each constraint. Experiments show that our algorithm outperforms the state of the art methods based on constraint techniques.

**Keywords** Universally quantified inequalities · Branch and bound · Interval analysis

## 1 Introduction

Many applications in engineering require verifying safety and performance conditions of a given system; for instance verifying for all electrical current in the interval

---

This paper is an extended version of a paper published at the SAC 2008 conference [15].

A. Goldsztejn (✉)  
LINA, University of Nantes, CNRS, Nantes, France  
e-mail: alexandre.goldsztejn@gmail.com

C. Michel · M. Rueher  
I3S-CNRS, University of Nice-Sophia Antipolis, Nice, France  
e-mail: cpjm@polytech.unice.fr

M. Rueher  
e-mail: rueher@polytech.unice.fr

$[i_{\min}, i_{\max}]$  and all possible resistor values in the interval  $r_0 \pm 5\%$  that the voltage across the resistor remains lower than a safety bound  $u_{\max}$ . In other words, we have to verify the satisfiability of

$$(\forall i \in [i_{\min}, i_{\max}]) (\forall r \in [0.95 r_0, 1.05 r_0]) (ri \leq u_{\max}). \tag{1}$$

In design problems, requirements are a bit more general. Instead of just verifying these conditions, we have to determine *values* of design variables that satisfy safety and performance conditions for all values of some uncertain physical data. For instance, in a satellite problem of [2], we want to determine the possible orbit for a new satellite which has to remain far enough from the other satellites. All these problems can be modeled in the framework of quantified constraint satisfaction problems (QCSPs). QCSPs arise naturally in numerous other applications (cf. [2, 13, 14, 20, 21, 29]).

In this paper, we are interested in a restricted form of QCSPs where existential quantifiers precede universal quantifiers:

$$\exists x \in \mathbf{x}, \forall y \in \mathbf{y}, c_1(x, y) \wedge \dots \wedge c_p(x, y), \tag{2}$$

where  $x = (x_1, \dots, x_n)$  and  $y = (y_1, \dots, y_m)$  denote vectors of variables,  $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$  and  $\mathbf{y} = (\mathbf{y}_1, \dots, \mathbf{y}_m)$  stand for vectors of intervals over continuous domains, and constraints  $c_i$  are inequalities of the form  $f_i(x, y) \leq 0$ . In addition the applications mentioned before, this class of QCSPs can tackle the design of robust controllers [10–12, 20, 25, 32].

In general, we do not only want to solve the decision problem (2) but we also want to find assignments of the existential variables that satisfy the constraints. That is to say, we consider  $\mathbf{x}$  as a search space where we try to find values  $x \in \mathbf{x}$  such that relation  $\forall y \in \mathbf{y}, c_1(x, y) \wedge \dots \wedge c_p(x, y)$  holds. Such a value  $x$  is called a solution<sup>1</sup> of (2). We define the solution set of (2) in the following way:

$$\text{Sol} := \left\{ x \in \mathbf{x} : \forall y \in \mathbf{y} \bigwedge_{i \in \{1, \dots, p\}} c_i(x, y) \right\}. \tag{3}$$

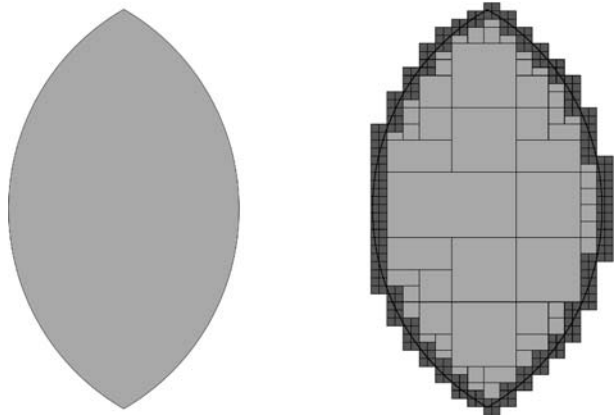
An essential observation is that in all above mentioned applications, the solutions are continua of real numbers. Indeed, the users are not really interested by isolated solutions but by continuous subsets of  $\mathbf{x}$  where all points are solutions. For instance, in design problems determining the real value of a physical value without any tolerance information does not really make sense. That is why we compute two sets  $\mathcal{I}$  and  $\mathcal{B}$  (respectively called the *inner approximation* and the *boundary approximation*) which satisfy the following relation:

$$\mathcal{I} \subseteq \text{Sol} \subseteq \mathcal{I} \cup \mathcal{B}. \tag{4}$$

Set  $\mathcal{I}$  contains only solutions whereas set  $\mathcal{B}$  contains the boundary of the solution set. So,  $\mathcal{B}$  contains both solutions and non-solutions.  $\mathcal{I} \cup \mathcal{B}$  contains the whole solution set and is called the *outer approximation*. Actually, due to computational limitations, we approximate  $\mathcal{I}$  and  $\mathcal{B}$  with union of boxes (cf. Fig. 1). Figure 2 provides an example of the output computed by our system.

<sup>1</sup>When several quantifier blocks alternate, the definition of solutions of the QCSP can be generalized to winning strategies [5, 6].

**Fig. 1** On the left hand side, in light gray, the solution set  $Sol$  with non null volume; on the right hand side, inner approximation  $\mathcal{I}$  in light gray and boundary approximation  $\mathcal{B}$  in dark gray. Note that  $\mathcal{I} \cup \mathcal{B} \supseteq Sol$



In this paper, we propose a new algorithm for computing the sets of boxes  $\mathcal{I}$  and  $\mathcal{B}$ . In contrast to other methods dedicated to the resolution of QCSPs, we handle the QCSP (2) as a classical CSP where the quantifiers are handled locally, at the level of each constraint: we introduce a CSP equivalent to (2) formed of constraints  $c^y(x)$  on the variables  $x$  (the domain parameter  $\mathbf{y}$  is explicitly given in exponent for clarity). These constraints are quantified inequalities:  $c^y(x) \equiv \forall y \in \mathbf{y}, f(x, y) \leq 0$ ;  $y$  are called parameters<sup>2</sup> and  $\mathbf{y}$  defines the variation domains of parameters  $y$ . We introduce a generic branch and prune algorithm dedicated to continuous CSPs with such parametric constraints. The handling of the parameter domains is an essential feature of our algorithm.

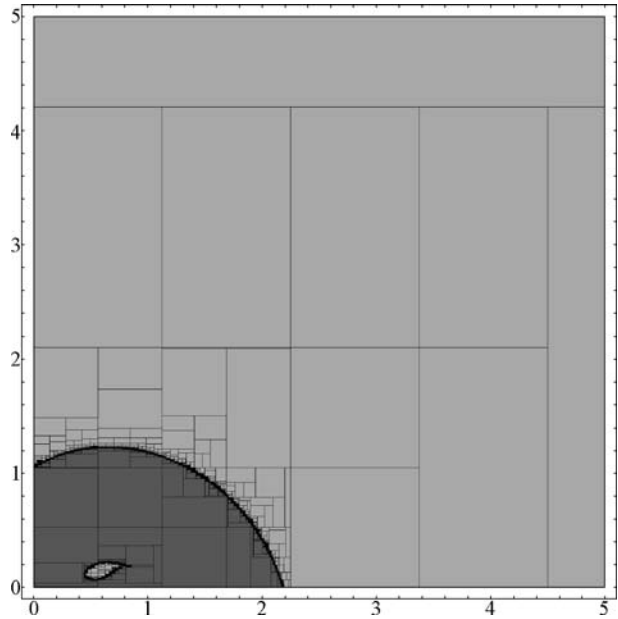
Other methods [1, 2, 27, 28] have been proposed to compute approximations of the solution set of (2). The new algorithm we introduce here is both much simpler and much more efficient than the previously proposed algorithms. First experiments shows that our algorithm outperforms the methods proposed in [1, 2, 27, 28].

**Notations** Boldface symbols denotes intervals, e.g.  $\mathbf{x} = [\underline{x}, \bar{x}] = \{x \in \mathbb{R} : \underline{x} \leq x \leq \bar{x}\}$  where  $\underline{x}$  and  $\bar{x}$  belongs to a finite subset of  $\mathbb{R}$ , usually the floating point numbers. The set of these intervals is denoted by  $\mathbb{IR}$  and the set of  $n$  dimensional interval vectors (also called boxes) by  $\mathbb{IR}^n$ . The width of an interval vector  $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$  is  $\text{wid}(\mathbf{x}) = \max_i |\underline{x}_i - \bar{x}_i|$ . The midpoint of an interval,  $(\bar{x} - \underline{x})/2$ , is denoted by  $\text{mid}(\mathbf{x})$ .  $\bar{x}$  denotes a value of  $\mathbf{x}$ . The interval hull of two boxes  $\mathbf{x}$  and  $\mathbf{y}$  is denoted by  $\square(\mathbf{x} \cup \mathbf{y})$  and is the smallest box containing both  $\mathbf{x}$  and  $\mathbf{y}$ . Note the difference between the union  $[-1, 1] \cup [2, 3] = \{x \in \mathbb{R} : -1 \leq x \leq 1 \vee 2 \leq x \leq 3\}$  which is disconnected, and the interval hull  $\square([-1, 1] \cup [2, 3]) = [-1, 3]$ . The set difference,  $\{x \in \mathbf{x} : x \notin \mathbf{y}\}$ , is denoted by  $\mathbf{x} \setminus \mathbf{y}$ .

Finally for reading convenience, if  $\mathcal{C}$  is a set of constraints then  $\mathcal{C}(x)$  denotes  $\forall c \in \mathcal{C}, c(x)$ , and  $\cup\{\mathbf{x}^1, \dots, \mathbf{x}^k\}$  denotes the union of these boxes, i.e.  $\mathbf{x}^1 \cup \dots \cup \mathbf{x}^k$ .

<sup>2</sup>We call  $y$  a parameter since universally quantified variables represent all the permissible values for a design parameter; in other words, the constraint must hold for any value inside the corresponding interval.

**Fig. 2** Inner (dark gray) and outer (dark gray and black) approximation of the solution set of problem robot (cf. [2]). Light gray are proved to contain no solution



**Outline of the paper** Section 2 recalls some basics on CSP with continuous domains. Section 3 describes the key features of the algorithm we propose. Experimental results are provided in Section 5.

**2 Basics on CSP with continuous domains**

To tackle CSP with continuous domains, a key issue is to be able to prove properties on continua of real numbers. Interval analysis handles this problem in an efficient way by using computations on floating point numbers.

We recall here some basics which are required to understand the paper. More details can be found in [17, 19, 22, 26].

An interval contractor for a constraint  $c$  with  $n$  variables is a function

$$\text{Contract}_c : \mathbb{IR}^n \longrightarrow \mathbb{IR}^n, \tag{5}$$

that satisfies the following relations:

1.  $\text{Contract}_c(\mathbf{x}) \subseteq \mathbf{x}$ ;
2.  $\forall x \in \mathbf{x} : c(x) \implies x \in \text{Contract}_c(\mathbf{x})$ .

Contractors for standard inequality constraints can be implemented using several techniques [3, 9, 23, 26]. Examples and experiments presented in this paper use contractors based on  $2B$ -consistency.

$2B$ -consistency is a relaxation of Arc-consistency [24], a concept which plays a key role in constraint programming.  $2B$ -consistency [4, 7, 23], also known as hull consistency, states a local property on the bounds of the domain of a variable at a single constraint level. More precisely, a constraint  $c$  is  $2B$ -consistent if, for any

variable  $x_i$ , there exist values in the domains of all other variables of  $c$  which satisfy  $c$  when  $x_i$  is fixed to either  $\underline{x}_i$  or  $\bar{x}_i$ .

Filtering by 2B-consistency of  $P = (\mathcal{X}, \mathcal{C}, \mathbf{x})$  produces a CSP  $P' = (\mathcal{X}, \mathcal{C}, \mathbf{x}')$  such that:

- $P$  and  $P'$  have the same solutions;
- $P'$  is 2B-consistent;
- $\mathbf{x}' \subseteq \mathbf{x}$  and the domains in  $\mathbf{x}'$  are the largest ones for which  $P'$  is 2B-consistent.

Filtering by 2B-consistency of  $P$  always exists and is unique [23].

### 3 Description of the algorithm

The key idea of our algorithm is to reformulate a QCSP as a CSP but with parametric constraints. More Precisely, we reformulate a QCSP with constraints of type (2) as a CSP  $\langle \mathcal{X}, \mathcal{C}, \mathbf{x} \rangle$  where:

- $\mathcal{X} = (x_1, \dots, x_n)$ ;
- $\mathcal{C} = \{c_1^y, \dots, c_p^y\}$  with  $c_i^y(x) \equiv \forall y \in \mathbf{y}, f_i(x, y) \leq 0$ ;
- $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ .

The parameter domains are considered at the level of each constraint. The advantage is that parameters can be handled in a way well suited to each constraint. Standard techniques have to be adapted for this class of constraints. For example, we show in Subsection 3.2 that  $\text{Contract}_{\forall y \in \mathbf{y}, f(x,y) \leq 0}(\mathbf{x})$  can be achieved with  $\text{Contract}_{f(x,\bar{y}) \leq 0}(\mathbf{x})$  for some arbitrary  $\bar{y} \in \mathbf{y}$ ; the latter operator being implemented using standard interval contractor.

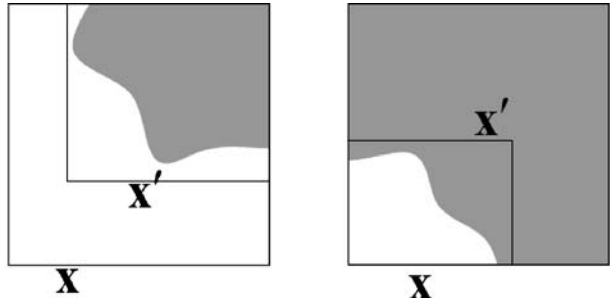
#### 3.1 Scheme of the algorithm

The branch and prune algorithm we propose alternates pruning and branching steps in a standard way to reject parts of the space that do not contain any solution. This process is interleaved with the identification of inner boxes that contain only solutions (see Algorithm 1). Before going into the details, let us explain in an informal way the main characteristics of Algorithm 1.

**Pruning the search space** In the pruning step, we reject parts of the search space that do not contain any solution. During the exploration of the search space, a box  $\mathbf{x}$  is contracted to a smaller box  $\mathbf{x}'$  without loosing any solution (see the left hand side diagram of Fig. 3). Parts of the search space where we can prove that no solution exists will be pruned after some branching steps. The implementation of the pruning step is detailed in Subsection 3.2.

**Identifying sets of solutions** An essential observation is that solution identification can be performed efficiently with a contraction operator. When we search for solutions in a box  $\mathbf{x}$ , we contract  $\mathbf{x}$  to  $\mathbf{x}'$  such that  $\mathbf{x} \setminus \mathbf{x}'$  only contains solutions of the initial constraint. In other words, we shrink  $\mathbf{x}$  by removing boxes where all points are solutions (see the right hand side diagram of Fig. 3). Again, solutions that cannot be identified by contracting  $\mathbf{x}$  will be identified after some branching steps. The solution set identification step is detailed in Subsection 3.3.

**Fig. 3** Pruning (left diagrams) and identification of solutions (right diagrams) both uses contractions of the domain



**Handling of parameter domains** The pruning and the identification of solutions sets are based on interval contractors. The point is that the efficiency of interval contractors increases as the size of the domains decreases. Therefore, an efficient algorithm must be able to work on subparts of the parameter domains while preserving the initial solution set. Our algorithm takes advantage of three different parameter handling techniques which are detailed in Subsection 3.4: *parameter domain pruning*, *parameter domain bisection* and *parameter instantiation*.

**Algorithm 1:** Generic Branch and Prune Algorithm

```

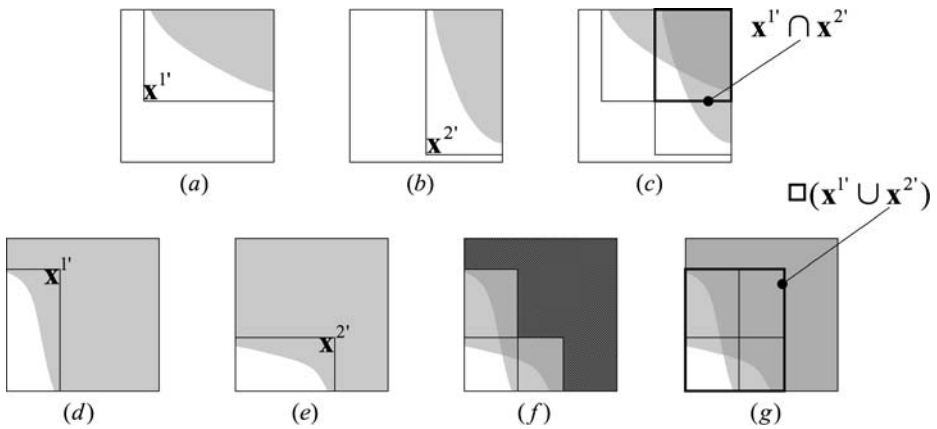
Input:  $\mathcal{C}, x, \epsilon$ 
Output:  $(\mathcal{I}, \mathcal{B})$ 
1  $\mathcal{U} \leftarrow \{(\mathcal{C}, x)\}; \mathcal{B} \leftarrow \{\}; \mathcal{I} \leftarrow \{\};$ 
2 while  $(\mathcal{U} \neq \emptyset)$  do
3    $(\mathcal{C}, x) \leftarrow \text{extract}(\mathcal{U});$ 
4   if  $\text{wid}(x) > \epsilon$  then
5      $\mathcal{C}' \leftarrow \text{ParameterInstantiation}(\mathcal{C}, x);$ 
6      $x' \leftarrow \text{Pruning}(\mathcal{C}', x);$ 
7      $(\mathcal{C}'', x'', \mathcal{I}') \leftarrow \text{SolutionIdentification}(\mathcal{C}', x');$ 
8      $\mathcal{C}''' \leftarrow \text{ParameterDomainBisection}(\mathcal{C}'', x'');$ 
9      $\{x''^1, x''^2\} \leftarrow \text{Branching}(x'');$ 
10     $\mathcal{U} \leftarrow \mathcal{U} \cup \{(x''^1, \mathcal{C}'''), (x''^2, \mathcal{C}''')\};$ 
11     $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{I}';$ 
12  else
13     $\mathcal{B} \leftarrow \mathcal{B} \cup \{x\};$ 
14  end
15 end
16 return  $(\mathcal{I}, \mathcal{B});$ 

```

Next subsections detail the implementation of the different functions.

3.2 Pruning

**Local Pruning** Given a constraint  $\forall y \in \mathbf{y}^i, c_i(x, y)$  and a box  $\mathbf{x}$ , we want to contract  $\mathbf{x}$  rejecting only parts that do not contain any solution of this constraint. To achieve this task, Benhamou et al. [2] and Ratschan [28] apply  $\text{Contract}_{c_i(x, \mathbf{y}^i)}(\mathbf{x})$ , where  $y$



**Fig. 4** Upper row pruning computed with local contractors. Lower row identification of solutions using local contractors on the negation of the constraints

is handled as an existentially quantified variable<sup>3</sup> in the domain  $\mathbf{y}^i$ . This strategy dramatically lacks of efficiency. That is why we propose a better contractor by instantiating the parameter to an arbitrary value  $\tilde{y} \in \mathbf{y}^i$  (see Example 1). More formally, the box  $\mathbf{x}$  is contracted using  $\text{Contract}_{c_i(x, \tilde{y})}(\mathbf{x})$  which rejects only parts of  $\mathbf{x}$  that do not satisfy  $c_i(x, \tilde{y})$ , and thus do not satisfy  $\forall y \in \mathbf{y}^i, c_i(x, y)$ .

Many strategies can be used to choose  $\tilde{y}$ . We chose  $\tilde{y} = \text{mid}(\mathbf{y}^i)$ . Our experiments have shown that searching for a better value of  $\tilde{y}$  is not worthwhile.<sup>4</sup> Note that the pruning used throughout the examples is based the ideal 2B consistency.

*Example 1* Let us consider the constraint  $c(x)$  defined by  $\forall y \in \mathbf{y}, f(x, y) \leq 0$  with  $f(x, y) = 10y - x - y^2, \mathbf{y} = [0, 1]$ , and  $\mathbf{x} = [0, 15]$ . The solution set of this very simple CSP is the interval  $[9, 15]$ . To reject values of  $\mathbf{x}$  that do not satisfy  $c(x)$ , we apply  $\text{Contract}_{f(x, 0.5) \leq 0}(\mathbf{x})$ , which reduces  $\mathbf{x}$  to  $\mathbf{x}' = [4.75, 15]$ . The method proposed in [2, 28] computes  $\text{Contract}_{f(x, y) \leq 0}(\mathbf{x})$  but cannot achieve any domain reduction.

**Global Pruning** The contraction of  $\mathbf{x}$  using one constraint removes boxes which do not contains any solutions for this constraint, and therefore non solutions boxes of the global CSP. This is illustrated in the first row of Fig. 4 where diagrams (a) and (b) show how the pruning computed with two local contractors.<sup>5</sup> leads to  $\mathbf{x}^{1'}$  and  $\mathbf{x}^{2'}$ .

<sup>3</sup>When  $y$  is handled as an existentially quantified variable, the pruning can lead to a reduction of the domain of  $y$ , hence proving that the universally quantified constraint is false on the whole domain of the variables. This inference is used in Ratschan [2, 28]. However, our experiments have shown that this reduction happens only in situations where the constraint can be easily proved to be false. Hence, the speedup is negligible in front of the other improvement we propose.

<sup>4</sup>On one hand, the cost of some derivative-based local search is too high w.r.t. the improvement of the contraction obtained with a better choice of  $\tilde{y}$ . On the other hand, Section 3.4.3 shows how to use derivatives in an efficient way, which replaces local search in a advantageous way (see Example 4 in Subsection 3.4).

<sup>5</sup>By local contractors, we mean contractors which only works at the level of a single constraint.

The global contraction depicted in Diagram (c) is obtained by computing  $\mathbf{x}^{1'} \cap \mathbf{x}^{2'}$ . Of course, in practice we compute this intersection in an incremental way.

### 3.3 Identification of sets of solution

As in [2, 8, 27, 28, 30, 31], the identification of sets of solutions is implemented by applying interval contractors to the negation of the constraints. Again, this process is achieved for each constraint separately. Only boxes that are proved to be local solutions for all constraints are solutions of the whole CSP.

**Identification of Solutions for a Single Constraint** In order to identify parts of a box  $\mathbf{x}$  containing only solutions of the constraint  $\forall y \in \mathbf{y}^i, f_i(x, y) \leq 0$ , we use the classical principle of pruning the negation of that constraint [8]. More precisely, we compute:

$$\langle \mathbf{x}^{i'}, \mathbf{y}^{i'} \rangle = \text{Contract}_{f_i(x,y) > 0}(\langle \mathbf{x}, \mathbf{y}^i \rangle), \tag{6}$$

where  $\langle \mathbf{x}, \mathbf{y} \rangle$  stands for the vector  $(\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{y}_1, \dots, \mathbf{y}_m)$ . Thus, every value of  $\mathbf{x} \setminus \mathbf{x}^{i'}$  satisfies  $f_i(x, y) \leq 0$  for all values of  $y$  in  $\mathbf{y}^i$  (cf. Fig. 5).

The following proposition obviously holds (see [2] and [27] for further details). Its detailed proof is provided here, as some parts of this proofs will be used in the proof of Proposition 2.

**Proposition 1** *Let  $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$  be a continuous function,  $\mathbf{x} \in \mathbb{I}\mathbb{R}^n, \mathbf{y} \in \mathbb{I}\mathbb{R}^m$  and  $c(x) \equiv \forall y \in \mathbf{y} f(x, y) \leq 0$ . Define*

$$\langle \mathbf{x}', \mathbf{y}' \rangle = \text{Contract}_{f(x,y) > 0}(\langle \mathbf{x}, \mathbf{y} \rangle). \tag{7}$$

*Then  $\forall x \in (\mathbf{x} \setminus \mathbf{x}'), \forall y \in \mathbf{y}, f(x, y) \leq 0$ , i.e.  $\mathbf{x} \setminus \mathbf{x}'$  is included in the constraint solution set.*

*Proof* By definition of a contractor,  $\forall x \forall y (x \in \mathbf{x} \wedge y \in \mathbf{y} \wedge f(x, y) > 0) \implies x \in \mathbf{x}' \wedge y \in \mathbf{y}'$ . The contrapositive of this implication also holds:

$$\forall x \forall y (x \notin \mathbf{x}' \vee y \notin \mathbf{y}') \implies x \notin \mathbf{x} \vee y \notin \mathbf{y} \vee f(x, y) \leq 0. \tag{8}$$

Now, since  $A \implies (\text{not} B) \vee C$  is equivalent to  $(A \wedge B) \implies C$ , we have

$$\forall x \forall y ((y \notin \mathbf{y}' \vee x \notin \mathbf{x}') \wedge x \in \mathbf{x} \wedge y \in \mathbf{y}) \implies f(x, y) \leq 0, \tag{9}$$

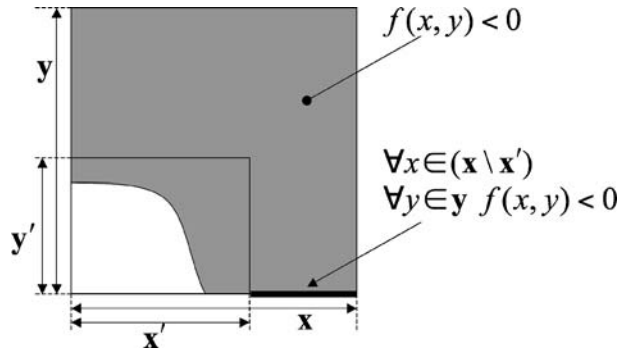
which trivially implies  $\forall x \forall y (x \notin \mathbf{x}' \wedge x \in \mathbf{x} \wedge y \in \mathbf{y}) \implies f(x, y) \leq 0$ . So,  $\mathbf{x} \setminus \mathbf{x}'$  contains only solutions of the constraint  $c(x)$ . □

Thus,  $\mathbf{x} \setminus \mathbf{x}'$  contains only solutions of the constraint  $c(x)$ .

*Example 2* Continuing Example 1, let us consider  $c(x)$  and  $\mathbf{x} = [4.75, 15]$ . In order to identify values of  $\mathbf{x}$  that satisfy  $c(x)$ , we apply  $\text{Contract}_{f(x,y) \geq 0}(\langle \mathbf{x}, \mathbf{y} \rangle)$  and obtain  $\mathbf{x}' = [4.75, 10]$  and  $\mathbf{y}' = [0.475, 1]$ . Therefore,  $\mathbf{x} \setminus \mathbf{x}' = ]10, 15]$  contains only solutions of the constraint.

**Identification of Solutions of the whole CSP** The boxes  $\mathbf{x}^{i'}$  for  $i \in \{1, \dots, p\}$  have been computed such that all values of  $\mathbf{x} \setminus \mathbf{x}^{i'}$  verify  $c_i(x)$  (cf. the diagrams (d) and

**Fig. 5** Local identification of solutions



(e) of Fig. 4). Thus, the values of  $\mathbf{x}$  that are outside all  $\mathbf{x}^{i'}$  satisfy all the constraints. Formally,

$$\mathbf{x} \setminus (\mathbf{x}^{1'} \cup \dots \cup \mathbf{x}^{p'}) \tag{10}$$

contains only solutions of the CSP. This inner approximation (10) is the dashed area of Diagram (f) of Fig. 4. The description of these inner approximations may become very complicated for higher dimensions and when numerous constraints are involved. That is why we use a weaker inner approximation

$$\mathbf{x} \setminus \square(\mathbf{x}^{1'} \cup \dots \cup \mathbf{x}^{p'}). \tag{11}$$

Inner approximation (11) is represented on Diagram (g) of Fig. 4. Let us define  $\mathbf{x}' = \square(\mathbf{x}^{1'} \cup \dots \cup \mathbf{x}^{p'})$ . The closure<sup>6</sup> of  $\mathbf{x} \setminus \mathbf{x}'$  is added to the set of inner boxes while  $\mathbf{x}'$  has to be further explored.

### 3.4 Handling parameter domains

As said before, the size of the domains of the parameters is a critical issue when applying interval contractors. This section details the three methods implemented in our algorithm. All these methods only apply interval contractors to parts of the initial parameter domains while keeping unchanged the CSP solution set.

#### 3.4.1 Parameter domain pruning

While computing (6), the domains of the variables  $\mathbf{x}$  are reduced to  $\mathbf{x}^{i'}$  and the domains of the parameters  $\mathbf{y}^i$  are reduced to  $\mathbf{y}^{i'}$  (cf. Fig. 5). In [2], the initial parameter domain  $\mathbf{y}^i$  is restored so this contraction of the parameter domains is not propagated. However, this reduced domain can be used while keeping unchanged the solution set of the constraint. In other words, showing that, after the identification of the solutions of a constraint, we can propagate the reduced parameter domains of  $c'(x)$  instead of the initial parameter domains of  $c(x)$ . Note that [28] also propagates this additional

<sup>6</sup>The set difference  $\mathbf{x} \setminus \mathbf{x}'$  is not closed in general, e.g.  $[-10, 10] \setminus [-1, 1] = [-10, -1] \cup [1, 10]$ . Nevertheless, we can observe that if  $f$  is continuous and non positive in  $\mathbf{x} \setminus \mathbf{x}'$ , then it is also non positive in its closure  $\text{cl}(\mathbf{x} \setminus \mathbf{x}')$ , e.g. if  $f(x) \leq 0$  in  $[-10, -1] \cup [1, 10]$  then  $f(x) \leq 0$  also holds in  $[-10, -1] \cup [1, 10]$ . Thus,  $\text{cl}(\mathbf{x} \setminus \mathbf{x}')$ , which can be described by a simple union of boxes, is recorded instead of  $\mathbf{x} \setminus \mathbf{x}'$ .

information, though this is not emphasized in [28]. It is hence worthwhile presenting and proving here this property directly.

**Proposition 2** *Let  $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$  be a continuous function,  $\mathbf{x} \in \mathbb{IR}^n$ ,  $\mathbf{y} \in \mathbb{IR}^m$  and  $c(x)$  the constraint  $\forall y \in \mathbf{y} \ f(x, y) \leq 0$ . Let*

$$\langle \mathbf{x}', \mathbf{y}' \rangle = \text{Contract}_{f(x,y)>0}(\langle \mathbf{x}, \mathbf{y} \rangle). \tag{12}$$

Then

$$x \in \mathbf{x} \wedge c(x) \iff (x \in \mathbf{x}' \wedge c'(x)) \vee x \in \mathbf{x} \setminus \mathbf{x}', \tag{13}$$

where  $c'(x) \equiv \forall y \in \mathbf{y}' \ f(x, y) \leq 0$ . So  $\mathbf{x} \setminus \mathbf{x}'$  is included in the constraint's solution set while  $c'(x)$  can be used instead of  $c(x)$  keeping unchanged the solution set.

*Proof*  $x \in \mathbf{x} \wedge c(x)$  is equivalent to  $(x \in \mathbf{x} \setminus \mathbf{x}' \vee x \in \mathbf{x}') \wedge c(x)$  and therefore is equivalent to  $(x \in \mathbf{x} \setminus \mathbf{x}' \wedge c(x)) \vee (x \in \mathbf{x}' \wedge c(x))$ . We know that  $c(x)$  is true for all  $x$  in  $\mathbf{x} \setminus \mathbf{x}'$ . As a consequence,  $x \in \mathbf{x} \setminus \mathbf{x}' \wedge c(x)$  can be reduced to  $x \in \mathbf{x} \setminus \mathbf{x}'$ . Moreover,  $x \in \mathbf{x}' \wedge c(x)$  is equivalent to

$$x \in \mathbf{x}' \wedge \forall y \in \mathbf{y}', f(x, y) \leq 0 \wedge \forall y \in \mathbf{y} \setminus \mathbf{y}', f(x, y) \leq 0. \tag{14}$$

Equation (9) trivially entails  $\forall x \in \mathbf{x}', \forall y \in \mathbf{y} \setminus \mathbf{y}', f(x, y) \leq 0$  (which is illustrated by Fig. 5). Therefore, (14) is equivalent to  $x \in \mathbf{x}' \wedge c'(x)$ . Thus,  $x \in \mathbf{x} \wedge c(x)$  is equivalent to  $x \in \mathbf{x} \setminus \mathbf{x}' \vee (x \in \mathbf{x}' \wedge c'(x))$ .  $\square$

Propagating parameter domains lead to a significant improvement and can be implemented without any overhead as parameter domains have to be contracted during the solution identification step.

### 3.4.2 Parameter domain bisection

Bisecting parameter domains is the most obvious way to reduce them and to ensure the convergence of the algorithm. To bisect parameter domains, we change the constraint  $\forall y \in \mathbf{y}, c(x, y)$  to the conjunction of the two constraints  $\forall y \in \mathbf{y}^1, c(x, y)$  and  $\forall y \in \mathbf{y}^2, c(x, y)$ , where  $\mathbf{y}^1$  and  $\mathbf{y}^2$  are obtained by bisecting  $\mathbf{y}$ . Example 3 illustrates how the parameter domain bisection is performed, and how it improves the pruning process as well as the identification of solutions.

*Example 3* Let us consider again Example 1 where the constraint store contains only one constraint  $\mathcal{C} = \{(\forall y \in \mathbf{y}, f(x, y) \leq 0)\}$ . Bisecting parameter domains, we obtain the new constraint store  $\mathcal{C}' = \{(\forall y \in \mathbf{y}^1, f(x, y) \leq 0), (\forall y \in \mathbf{y}^2, f(x, y) \leq 0)\}$  with  $\mathbf{y}^1 = [0, 0.5]$  and  $\mathbf{y}^2 = [0.5, 1]$ . The latter constraint store is equivalent to the original one, but contains more constraints with smaller parameter domains.

The pruning operator is now applied to each constraint of the store, thus the two contractions  $\text{Contract}_{f(x,0.25) \leq 0}(\mathbf{x})$  and  $\text{Contract}_{f(x,0.75) \leq 0}(\mathbf{x})$  are computed. This leads to  $\mathbf{x}' = [6.9375, 15]$ . The pruning achieved here is sharper than the one computed without bisecting the parameter domains (cf. Example 1).

To identify solutions of this new CSP, we apply the contractor  $\langle \mathbf{x}^i, \mathbf{y}^i \rangle = \text{Contract}_{f(x,y) \geq 0}(\langle \mathbf{x}', \mathbf{y}^i \rangle)$  for  $i \in \{1, 2\}$ . We obtain  $\mathbf{x}^{1'} = \emptyset, \mathbf{y}^{1'} = \emptyset, \mathbf{x}^{2'} = [6.9375, 9.75]$  and  $\mathbf{y}^{2'} = [0.71875, 1]$ . Finally, the domain  $\mathbf{x}'$  is contracted to  $\square(\mathbf{x}^{1'} \cup \mathbf{x}^{2'}) =$

[6.9375, 9.75] while  $\text{cl}([6.9375, 15] \setminus [6.9375, 9.75]) = [9.75, 15]$  is proved to be an inner box. The solution identification is also more efficient thanks to the bisection of the parameter domain (cf. Example 2).

Bisection of parameter domains is used in [28] though through a different implementation.

Bisecting each constraint parameters domains multiplies by two the number of constraints in the store. We use the following heuristic to decide whether or not a constraint parameters domains should be bisected:

A quantified constraint  $(\forall y \in \mathbf{y}, f(x, y) \leq 0)$  may be bisected in two constraints  $(\forall y \in \mathbf{y}^1, f(x, y) \leq 0)$  and  $(\forall y \in \mathbf{y}^2, f(x, y) \leq 0)$ . The reason for using the last two constraints instead of the first one is that the contractor may be much more efficient on small domains. That is why we test whether bisecting parameter domains actually reduces the pessimism of interval evaluations. Hence, we compute the three following interval evaluations:  $\mathbf{z} := f(\mathbf{x}, \mathbf{y})$ ,  $\mathbf{z}^1 := f(\mathbf{x}, \mathbf{y}^1)$  and  $\mathbf{z}^2 := f(\mathbf{x}, \mathbf{y}^2)$  knowing that  $\square(\mathbf{z}^1 \cup \mathbf{z}^2) \subseteq \mathbf{z}$ . Usually this inclusion is strict, meaning that bisecting the domains actually decreases the pessimism of the interval evaluations. The heuristic we propose is to define a threshold on the ratio

$$\frac{\text{wid}(\square(\mathbf{z}^1 \cup \mathbf{z}^2))}{\text{wid}\mathbf{z}} \tag{15}$$

over which the parameters domains are not bisected. Such a decision relies on three interval evaluations of the function  $f$ , which turns out to be cheap w.r.t. the use of interval contractors. Our experiments have been carried out with a threshold of 0.8, which has shown to lead to good performances in average.

### 3.4.3 Parameter instantiation

The constraint  $\forall y \in \mathbf{y}, f(x, y) \leq 0$ , where  $\mathbf{y} = [\underline{y}, \bar{y}]$ , can be simplified if the function  $f$  is proved to be monotonic w.r.t. the parameter. Indeed, if  $f$  is increasing (resp. decreasing) w.r.t.  $y$ , then the quantified constraint is obviously equivalent to the non-quantified constraint  $f(x, \bar{y}) \leq 0$  (resp.  $f(x, \underline{y}) \leq 0$ ). The variation of the function can be checked by evaluating its derivative w.r.t. the parameter on the intervals  $\mathbf{x}$  and  $\mathbf{y}$ . The same property holds if there are several parameters, using a derivative w.r.t. each parameter. This is formalized by the following proposition:

**Proposition 3** *Let  $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$  be a differentiable function,  $\mathbf{x} \in \mathbb{I}\mathbb{R}^n$ ,  $\mathbf{y} \in \mathbb{I}\mathbb{R}^m$  and  $c(x) \equiv \forall y \in \mathbf{y} f(x, y) \leq 0$ . Let us also define  $\mathbf{y}'$  in the following way:*

$$\mathbf{y}'_i = \begin{cases} \inf \mathbf{y}'_i & \text{if } (\partial f / \partial y_i)(\mathbf{x}, \mathbf{y}) \leq 0 \\ \sup \mathbf{y}'_i & \text{if } (\partial f / \partial y_i)(\mathbf{x}, \mathbf{y}) \geq 0 \\ \mathbf{y}_i & \text{otherwise} \end{cases}, \tag{16}$$

where  $(\partial f / \partial y_i)(\mathbf{x}, \mathbf{y})$  is an interval extension of  $\partial f / \partial y_i$ . Then,  $\forall x \in \mathbf{x}, \forall y \in \mathbf{y}, f(x, y) \leq 0$  is equivalent to  $\forall y \in \mathbf{y}', f(x, y) \leq 0$ .

*Proof* Let us select an arbitrary  $x \in \mathbf{x}$ . As  $\mathbf{y}' \subseteq \mathbf{y}$ , proposition  $\forall y \in \mathbf{y}, f(x, y) \leq 0$  trivially implies  $\forall y \in \mathbf{y}', f(x, y) \leq 0$ . For the converse, we assume that  $\forall y \in \mathbf{y}', f(x, y) \leq 0$  holds and have to prove that  $\forall y \in \mathbf{y}, f(x, y) \leq 0$  holds. Without loss of generality we suppose that  $\mathbf{y}' = (\mathbf{y}_1, \dots, \mathbf{y}_k, \inf \mathbf{y}_{k+1}, \dots, \inf \mathbf{y}_n)$  (the case where  $f$  is increasing

w.r.t. some components is similar). Let us define  $\mathbf{y}'^j = (\mathbf{y}_1, \dots, \mathbf{y}_j, \inf \mathbf{y}_{j+1}, \dots, \inf \mathbf{y}_n)$  for  $k \leq j \leq n$ . Note that  $\mathbf{y}'^k = \mathbf{y}'$  and  $\mathbf{y}'^n = \mathbf{y}$ . We prove that  $\forall y \in \mathbf{y}'^j, f(x, y) \leq 0$  holds for all  $j \in \{k, \dots, n\}$  by induction on  $j$ . The base case is  $\forall y \in \mathbf{y}'^k, f(x, y) \leq 0$  which trivially holds. Now, we assume that  $\forall y \in \mathbf{y}'^j, f(x, y) \leq 0$  holds for some  $j < n$  and prove that  $\forall y \in \mathbf{y}'^{j+1}, f(x, y) \leq 0$  holds. Let  $y = (y_1, \dots, y_{j+1}, \inf \mathbf{y}_{j+2}, \dots, \inf \mathbf{y}_n)$  be an arbitrary vector of  $\mathbf{y}'^{j+1}$ . As  $f(y_1, \dots, y_{j+1}, \inf \mathbf{y}_{j+1}, \dots, \inf \mathbf{y}_n) \leq 0$  by induction hypothesis while  $f$  is decreasing w.r.t.  $y_{j+1}$ , we have  $f(y) \leq 0$ . Therefore,  $\forall y \in \mathbf{y}'^{j+1}, f(x, y) \leq 0$  holds.  $\square$

*Example 4* Let us come back to Example 1. Evaluating  $(\partial f / \partial y)(\mathbf{x}, \mathbf{y}) = 10 - 2y = [8, 10]$ , we prove that  $f$  is increasing w.r.t.  $y$ . Therefore, the constraint  $\forall y \in \mathbf{y}, f(x, y) \leq 0$  is equivalent to  $f(x, 1) \leq 0$ . Then, the pruning contracts  $\mathbf{x} = [0, 15]$  to  $[9, 15]$  and the solution identification proves that  $[9, 15]$  contains only solutions. On this simple example, we obtain an exact description of the solution set. This set is much sharper than the contractions obtained without this instantiation of the parameter (cf. Examples 1 and 2).

Parameter instantiation can drastically improve the efficiency of the pruning and the solution identification steps. However, evaluating derivatives may be expensive, in particular when they involve trigonometric function. Experiments presented in the next section illustrate well this issue.

### 4 Formal properties and correctness proof of Algorithm 1

Subsection 4.1 explains what the invariant of Algorithm 1 is, while Subsection 4.2 provides a formal property of each function. Finally the correctness of Algorithm 1 is proved in Subsection 4.3.

#### 4.1 Basic properties

Let  $\mathbf{x}_0$  and  $\mathcal{C}_0$  be the initial domains and the initial constraint set of the CSP. The while loop maintains the following two properties on the set  $\mathcal{U} = \{(\mathbf{x}_1, \mathcal{C}_1), \dots, (\mathbf{x}_q, \mathcal{C}_q)\}$  and the sets of boxes  $\mathcal{I}$  and  $\mathcal{B}$ :

$$x \in \mathbf{x}_0 \wedge \mathcal{C}_0(x) \implies (x \in \mathbf{x}_1 \wedge \mathcal{C}_1(x)) \vee \dots \vee (x \in \mathbf{x}_q \wedge \mathcal{C}_q(x)) \vee x \in \cup \mathcal{I} \vee x \in \cup \mathcal{B} \tag{17}$$

and

$$x \in \cup \mathcal{I} \implies \mathcal{C}_0(x). \tag{18}$$

These properties obviously hold after the initialization performed in Line 1. At the end of the algorithm, the set  $\mathcal{U}$  is empty while  $\mathcal{I}$  and  $\mathcal{B}$  have been filled. Letting  $\mathcal{U} = \emptyset$  in (17), we obtain the following relation:

$$x \in \mathbf{x}_0 \wedge \mathcal{C}_0(x) \implies x \in \cup \mathcal{I} \vee x \in \cup \mathcal{B}. \tag{19}$$

This means that  $\mathcal{I} \cup \mathcal{B}$  is an outer approximation of  $\{x \in \mathbf{x} : \mathcal{C}(x)\}$ . In other words, it states that we do not lose any solution. Relation (18) states that  $\mathcal{I}$  is an inner approximation.

The formal properties of each function are given in the next subsection.

### 4.2 Properties of the functions

Function *ParameterInstantiation()* returns a set of constraints  $\mathcal{C}'$  which is equivalent to  $\mathcal{C}$  for domain  $\mathbf{x}$ , but where some parameters have been instantiated (see Subsection 3.4.3). Formally, this function ensures that

$$x \in \mathbf{x} \wedge \mathcal{C}(x) \iff x \in \mathbf{x} \wedge \mathcal{C}'(x) \tag{20}$$

(see Proposition 3).

Function *Pruning()* contracts the domain  $\mathbf{x}$  without losing any solution of  $\mathcal{C}'$  (see Subsection 3.2). Therefore  $x \in \mathbf{x} \wedge \mathcal{C}'(x) \iff x \in \mathbf{x}' \wedge \mathcal{C}'(x)$ , which together with (20) yields

$$x \in \mathbf{x} \wedge \mathcal{C}(x) \iff x \in \mathbf{x}' \wedge \mathcal{C}'(x). \tag{21}$$

Function *SolutionIdentification()* contracts  $\mathbf{x}'$  to  $\mathbf{x}''$  and returns a list of inner boxes  $\mathcal{I}'$  (see Subsection 3.3). The domains of the parameters are also updated in  $\mathcal{C}''$  (see Subsection 3.4.1). Formally, this function ensures

$$x \in \mathbf{x}' \wedge \mathcal{C}'(x) \iff (x \in \mathbf{x}'' \wedge \mathcal{C}''(x)) \vee x \in \cup \mathcal{I}' \tag{22}$$

(see Proposition 2). Using (21) we obtain

$$x \in \mathbf{x} \wedge \mathcal{C}(x) \iff (x \in \mathbf{x}'' \wedge \mathcal{C}''(x)) \vee x \in \cup \mathcal{I}', \tag{23}$$

from which trivially follows that  $\mathcal{I}'$  contains only solutions, i.e.

$$x \in \cup \mathcal{I}' \implies \mathcal{C}(x). \tag{24}$$

Function *ParameterDomainBisection()* bisects the parameter domains of the constraints and thus increases the number of constraints in  $\mathcal{C}''$  (see Subsection 3.4.2). Formally, this functions ensures that  $x \in \mathbf{x}'' \wedge \mathcal{C}''(x)$  is equivalent to  $x \in \mathbf{x}''' \wedge \mathcal{C}'''(x)$ , which together with (23) yields

$$x \in \mathbf{x} \wedge \mathcal{C}(x) \iff (x \in \mathbf{x}''' \wedge \mathcal{C}'''(x)) \vee x \in \cup \mathcal{I}'. \tag{25}$$

Function *Branching()* achieves a standard bisection of the domains  $\mathbf{x}''$ .

### 4.3 Correctness of the algorithm

Algorithm 1 is correct if properties (17) and (18) hold at the end of the execution, i.e., if  $\mathcal{I}$  contains only solutions while every solution is contained in  $\mathcal{I} \cup \mathcal{B}$ . The correctness proof of Algorithm 1 is done by induction. Properties (17) and (18) hold after initialization at Line 1. To prove the overall correctness of our algorithm, we just have to prove that (17) and (18) are maintained by the *while* loop.

Let  $\mathcal{U} = \{(\mathbf{x}_1, \mathcal{C}_1), \dots, (\mathbf{x}_q, \mathcal{C}_q)\}$ ,  $\mathcal{I} = \{\mathbf{i}_1, \dots, \mathbf{i}_s\}$  and  $\mathcal{B} = \{\mathbf{b}_1, \dots, \mathbf{b}_t\}$ . Before the execution of the *while* loop, we assume by induction hypothesis that  $\mathcal{U}$ ,  $\mathcal{I}$  and  $\mathcal{B}$  satisfy (17) and (18). Let us denote these sets by  $\tilde{\mathcal{U}}$ ,  $\tilde{\mathcal{I}}$  and  $\tilde{\mathcal{B}}$  respectively. After execution of the *while* loop, without loss of generality, the element extracted in Line

3 is the first of  $\mathcal{U}$ , i.e.  $(\mathbf{x}, \mathcal{C}) = (\mathbf{x}_1, \mathcal{C}_1)$  and  $\tilde{\mathcal{U}} = \{(\mathbf{x}_2, \mathcal{C}_2), \dots, (\mathbf{x}_q, \mathcal{C}_q)\}$ . We just have to prove that  $\tilde{\mathcal{U}}, \tilde{\mathcal{I}}$  and  $\tilde{\mathcal{B}}$  satisfy (17) and (18). First of all, note that

$$x \in \mathbf{x} \wedge \mathcal{C}(x) \implies \mathcal{C}_0(x) \tag{26}$$

is a trivial consequence of (17). Now, consider the two cases of the *if* statement on Line 4:

1. Suppose that  $\text{wid}(\mathbf{x}) \leq \epsilon$ , so the *else* statement is executed. Then  $\tilde{\mathcal{I}} = \mathcal{I}$  and  $\tilde{\mathcal{B}} = \{\mathbf{b}_1, \dots, \mathbf{b}_t, \mathbf{x}_1\}$ . Therefore, we have to prove that both  $x \in \mathbf{x}_0 \wedge \mathcal{C}_0(x)$  implies

$$(x \in \mathbf{x}_2 \wedge \mathcal{C}_2(x)) \vee \dots \vee (x \in \mathbf{x}_q \wedge \mathcal{C}_q(x)) \vee x \in \cup \mathcal{I} \vee x \in \mathbf{b}_1 \cup \dots \cup \mathbf{b}_t \cup \mathbf{x}_1 \tag{27}$$

and  $x \in \cup \mathcal{I} \implies \mathcal{C}_0(x)$  hold. Both are trivial consequences of the induction hypotheses (17) and (18).

2. Suppose that  $\text{wid}(\mathbf{x}) > \epsilon$ , so the *then* statement is executed. Then  $\tilde{\mathcal{U}} = \{(\mathbf{x}_2, \mathcal{C}_2), \dots, (\mathbf{x}_q, \mathcal{C}_q), (\mathbf{x}''^1, \mathcal{C}''^1), (\mathbf{x}''^2, \mathcal{C}''^2)\}$ ,  $\tilde{\mathcal{I}} = \mathcal{I} \cup \mathcal{I}'$  and  $\tilde{\mathcal{B}} = \mathcal{B}$ . From the induction hypothesis (18) and relations (24) and (26), we obtain  $x \in \cup \tilde{\mathcal{I}} \implies \mathcal{C}_0(x)$ . From the induction hypothesis (17) and relations (25) and (26),  $x \in \mathbf{x}_0 \wedge \mathcal{C}_0(x)$  implies

$$(x \in \mathbf{x}_2 \wedge \mathcal{C}_2(x)) \vee \dots \vee (x \in \mathbf{x}_q \wedge \mathcal{C}_q(x)) \vee (x \in \mathbf{x}'' \wedge \mathcal{C}''(x)) \vee x \in \cup \tilde{\mathcal{I}} \vee x \in \cup \mathcal{B}. \tag{28}$$

Since  $\mathbf{x}'' = \mathbf{x}''^1 \cup \mathbf{x}''^2$ , (28) entails

$$\begin{aligned} &(x \in \mathbf{x}_2 \wedge \mathcal{C}_2(x)) \vee \dots \vee (x \in \mathbf{x}_q \wedge \mathcal{C}_q(x)) \vee (x \in \mathbf{x}''^1 \wedge \mathcal{C}''^1(x)) \\ &\vee (x \in \mathbf{x}''^2 \wedge \mathcal{C}''^2(x)) \vee x \in \cup \tilde{\mathcal{I}} \vee x \in \cup \mathcal{B}. \end{aligned} \tag{29}$$

This proves the correctness of our algorithm.

### 5 Experiments

This section compares the results of the IPA system described in [2], Rsolver [28] with our system, called “Qine”, on a set of 9 benchmarks.

The 9 benchmarks are coming from the literature. The Circle, PathPoint, Parabola, Robot and Satellite QCSPs are taken from [2]. A description of Robust1 (respectively, Robust4, Robust5 and Robust6) can be found in [10] (respectively, [20, 25] and [18]).

All benchmarks have been run on an Intel Core Duo 2 at 2.4Ghz with a time out of 600s. To limit the effect of the high memory consumption of these algorithms, the available memory has been restricted to 1Gb. Thus, a benchmark could either succeed to run within these two limits, ends with a time out (“-”), or reach the memory limit (“-\*”).

Table 1 reports the results obtained with Rsolver and Qine. It gives Rsolver timing, as well as, the time required to solve the benchmarks for the different Qine running options:

- “2B” is the implementation of our own algorithm and relies on the contractor derived from 2b-consistency. More precisely, its implementation relies on a

**Table 1** Timing (in seconds) for Rsolver vs Qine

Problem	Ratio	Rsolver	R2B	R2B+	Qine	
					2B	2B+
Circle	0.98	55.67	8.35	4.04	0.90	1.03
	0.99	169.51	23.59	8.09	2.32	2.22
	0.999	–	–	100.76	69.18	31.55
PathPoint	0.6	122.72	–	–	0.01	0.01
	0.65	334.34	–	–	0.01	0.02
	0.7	–	–	–	0.02	0.03
Parabola	0.96	48.33	12.42	2.44	1.47	1.36
	0.97	130.30	38.50	3.47	2.74	1.92
	0.98	–	85.43	8.34	8.77	6.59
Robot	0.98	10.34	4.32	3.04	0.06	0.08
	0.99	26.74	14.58	4.32	0.14	0.18
	0.999	–	–	25.34	5.37	4.08
Satellite	0.5	303.30	75.79	119.10	71.36	114.33
	0.55	–	155.05	244.90	168.32	268.36
	0.6	–	273.89	433.95	227.09	368.90
Robust1	0.999	1.10	0.28	0.00	0.01	0.00
	0.9999	7.16	1.86	0.00	0.04	0.00
	0.99999	76.25	28.36	0.00	0.10	0.00
Robust4	0.5	–	–	0.01	–*	0.00
	0.55	–	–	0.01	–	0.01
	0.6	–	–	0.01	–	0.01
Robust5	0.7	75.35	0.00	0.00	0.00	0.00
	0.75	80.62	0.03	0.00	0.00	0.00
	0.8	–	0.46	0.00	0.01	0.00
Robust6	0.7	59.98	2.22	0.00	0.02	0.00
	0.75	224.33	18.94	0.00	0.05	0.00
	0.8	–	68.46	0.00	0.09	0.00

forward–backward evaluation<sup>7</sup> of the direct acyclic graph which represents the constraint.

- “2B+” combines the previous contractor with the derivative based parameter handling strategy introduced in Subsection 3.4.3.
- “R2B” is our implementation of Ratschan’s algorithm [28] within Qine. It mainly differs from Ratschan’s implementation by the use of our own contractors as well as by its capability to handle the class of constraints addressed in this paper. In order to explain the gain of our algorithm, we have added to Ratschan’s algorithm the derivative based parameter instantiation of Section 3.4.3. The benchmarks have been solved as well with this new algorithm, called “R2B+”. Thus, the only significant difference between “R2B+” and Qine 2B+ is the heuristic for bisecting the domains of the parameters.

<sup>7</sup>Contrary to [16], our implementation of the forward–backward evaluation does not produce nor propagate unions of intervals, i.e., the forward–backward evaluation is used here as a mean to implement a simple 2*b*-consistency.

In all the previous cases, the main branch and bound loop is stopped once the following ratio (column 2) have been reached:

$$\text{ratio} = (V_{\text{inner}} + V_{\text{outer}}) / V_{\text{initial}} \quad (30)$$

where  $V_{\text{inner}}$  is the total volume of the inner boxes,  $V_{\text{outer}}$  is the total volume of the outer boxes and  $V_{\text{initial}}$  is the volume given by the initial domains of the variables. Though all the benchmarks have been ran for ratios going from 0.5 to 0.99999, the table provides only the most significant results for the sake of space.

As shown in Table 1, in average, Qine outperforms Rsolver by one order of magnitude. For instance, Qine handles the Robust benchmark in a negligible time while Rsolver requires between 1.10 and 2.24s to solve these benchmarks and runs out of time for five of these benchmarks. This behavior is confirmed by our own implementation of Rsolver: though, thanks to the use of our own contractors, this implementation performs better than the original Rsolver implementation, it behaves worst when the ratio is raised. A comparison of the different available combinations shows that the “2B+” combination has a more robust behavior. It takes advantage of all the available informations and offers a good trade-off between the computation time and the domain reductions. However, on the Satellite bench, 2B performs better than 2B+. This illustrates the trade-off between the cost of derivative computation and the benefit of parameter instantiations.

The execution time required by R2B on the different benchmarks show that the gain of performance does not only come from our contractors: our contractors are obviously better than the one used in Rsolver but Qine, which uses the same contractor than R2B, is still much more efficient than R2B on numerous benchmarks. To better understand where the gain comes from, we have implemented in R2B+ the derivative based parameter instantiation. Execution times of R2B clearly show that the gains on the Robustk problems are related to this later heuristic. However, on the remaining problems, Qine 2B+ remains more efficient. This is due to the bisection heuristic for the parameters domains.

Table 2 compares IPA with Qine. Here again, Qine outperforms IPA. Indeed, IPA was able to solve the PathPoint bench within the timeout for only one of the tested ratios. However, IPA is faster than Qine on the Satellite bench for the lower ratio

**Table 2** Timing (in seconds) for IPA vs Qine

Problem	Ratio	IPA	QINE	
			2B	2B+
Circle	0.892	2.18	0.14	0.20
	0.932	8.09	0.20	0.30
	0.953	29.63	0.30	0.42
PathPoint	0.817	88.94	0.08	0.12
Parabola	0.871	0.34	0.06	0.06
	0.934	1.88	0.34	0.33
	0.965	29.55	1.80	1.65
Robot	0.992	1.32	0.26	0.34
	0.997	6.88	1.07	1.10
	0.998	17.53	1.89	1.81
Satellite	0.264	5.71	10.52	15.67
	0.437	33.61	25.10	37.09
	0.573	224.65	109.74	161.95

values. This behavior is probably due to the limit of the 2B based contractor whose domain reduction capabilities decrease when a variable has multiple occurrences within one constraint. IPA is based on a Box contractor which does not suffer from the same behavior (see [9] for a detailed comparison of 2B and Box). However, when the ratio value increase, Qine becomes faster than IPA.

Note that the class of QCSPs handled by IPA is limited to one parameter. Thus, IPA is not able to solve the Robust benchmark.

## 6 Conclusion

In this paper, we have introduced a simple and efficient algorithm to handle QCSPs formed of universally quantified inequalities over continuous domains. Examples coming from the literature reveal that this class covers numerous important practical applications, including the design of robust controllers.

The size of the domains of the universally quantified parameters is a critical issue in solving such QCSPs. Bisectioning these domains is required to ensure convergence. The simplicity of our algorithm is due to a specific parameter domain bisection technique: the parameter domain  $\mathbf{y}$  of the quantified constraint  $(\forall y \in \mathbf{y}, f(x, y) \leq 0)$  is bisected storing in the constraint store both  $(\forall y \in \mathbf{y}_1, f(x, y) \leq 0)$  and  $(\forall y \in \mathbf{y}_2, f(x, y) \leq 0)$ , where  $\mathbf{y}_1$  and  $\mathbf{y}_2$  are obtained bisecting  $\mathbf{y}$ . Another contribution concerns the instantiation of parameters to some specific values of their domains to allow the usage of effective techniques dedicated to non quantified CSPs.

We have compared our algorithm with the two state of the art algorithms which deal with quantified inequalities: IPA, from Benhamou et al., and Rsolver from Ratschan. IPA handles a smaller class of constraints than our algorithm, while Rsolver solves a larger class of constraints as it theoretically handles any form of quantified inequalities. First experiments show that our algorithm outperforms IPA, in spite of handling a larger class of constraints, and Rsolver. To obtain a more accurate comparison with the method used in Rsolver, we have implemented its algorithm in a optimized way for the restricted class of constraints we handle while relying on our own consistency techniques. Our method shows drastic improvement w.r.t. both Rsolver and our optimized implementation of its algorithm.

Further work concerns the improvement of algorithm including the most recent advances in filtering techniques. Techniques introduced in this paper, like the bisection of parameter domains or the derivative based parameter instantiation, may be used in a wider scope of application areas. In particular, we will study how the latter can be adapted to handle the a wider class of constraints.

**Acknowledgements** We are grateful to Marc Christie and Stefan Ratschan for their valuable help in the experiments. We are also grateful to anonymous referees whose comments greatly helped to improve this paper.

## References

1. Benhamou, F., & Goualard, F. (2000). Universally quantified interval constraints. In *Proceedings of international conference on principles and practice of constraint programming, LNCS*, (Vol. 1894, pp. 67–82).

2. Benhamou, F., Goualard, F., Languenou, E., & Christie, M. (2004). Interval constraint solving for camera control and motion planning. *ACM Transactions on Computational Logic*, 5(4), 732–767.
3. Benhamou, F., McAllester, D. A., & Van Hentenryck, P. (1994). CLP (intervals) revisited. In *SLP* (pp. 124–138).
4. Benhamou, F., & Older, W. (1997). Applying interval arithmetic to real, integer and Boolean constraints. *Journal of Logic Programming*, 6, 1–24.
5. Boerner, F., Bulatov, A., Chen, H., Jeavons, P., & Krokhin, A. (2003). The complexity of constraint satisfaction games and qcsp. In *Proc. of computer science logic, LNCS*, (Vol. 2803/2003, pp. 58–70).
6. Bordeaux, L., Cadoli, M., & Mancini, T. (2005). Csp properties for quantified constraints: Definitions and complexity. In *Proc. of amer. conf. on artificial intelligence (AAAI)* (pp. 360–365).
7. Cleary, J. G. (1987). Logical arithmetic. *Future Computing Systems* (pp. 125–149).
8. Collavizza, H., Delobel, F., & Rueher, M. (1999). Extending consistent domains of numeric CSP. In *Proceedings of IJCAI 1999*.
9. Collavizza, H., Delobel, F., & Rueher, M. (1999). Comparing partial consistencies. *Reliable Computing*, 1, 1–16.
10. Dorato, P. (2000). Quantified multivariate polynomial inequalities. *IEEE Control Systems Magazine*, 20(5), 48–58.
11. Dorato, P., Yang, W., & Abdallah, C. (1997). Robust multi-objective feedback design by quantifier elimination. *Journal of Symbolic Computations*, 24, 153–159.
12. Fiorio, G., Malan, S., Milanese, M., & Taragna, M. (1993). Robust performance design of fixed structure controllers for systems with uncertain parameters. In *Proceedings of the 32st IEEE conference on decision and control* (Vol. 4, pp. 3029–3031).
13. Goldsztejn, A. (2006) A branch and prune algorithm for the approximation of non-linear AE-resolution sets. In *SAC '06: Proceedings of the 2006 ACM symposium on applied computing* (pp. 1650–1654).
14. Goldsztejn, A., & Jaulin, L. (2006). Inner and outer approximations of existentially quantified equality constraints. In *Proceedings of CP 2006, LNCS*, (Vol. 4204/2006, pp. 198–212).
15. Goldsztejn, A., Michel, C., & Rueher, M. (2008). An efficient algorithm for a sharp approximation of universally quantified inequalities. In *Proceedings of ACM SAC 2008*. Fortaleza, Brazil.
16. Goualard, F., & Granvilliers, L. (2005). Controlled propagation in continuous numerical constraint networks. In *Proceedings of the 2005 ACM symposium on applied computing* (pp. 377–382).
17. Hayes, B. (2003). A lucid interval. *American Scientist*, 91(6), 484–488.
18. Jaulin, L., Braems, I., & Walter, E. (2002). Interval methods for nonlinear identification and robust control. In *In Proceedings of the 41st IEEE conference on decision and control* (Vol. 4, pp. 4676–4681).
19. Jaulin, L., Kieffer, M., Didrit, O., & Walter, E. (2001). *Applied interval analysis with examples in parameter and state estimation, robust control and robotics*. Springer-Verlag.
20. Jaulin, L., & Walter, E. (1996). Guaranteed tuning, with application to robust control and motion planning. *Automatica*, 32(8), 1217–1221.
21. Jirstrand, M. (1997). Nonlinear control system design by quantifier elimination. *Journal of Symbolic Computation*, 24(2), 137–152.
22. Kearfott, R. B. (1996). Interval computations: Introduction, uses, & resources. *Euromath Bulletin*, 2(1), 95–112.
23. Lhomme, O. (1993). Consistency techniques for numeric CSPs. In *Proceedings of IJCAI 1993* (pp. 232–238).
24. Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8, 99–118.
25. Malan, S., Milanese, M., & Taragna, M. (1997). Robust analysis and design of control systems using interval arithmetic. *Automatica*, 33(7), 1363–1372.
26. Neumaier, A. (1990). *Interval methods for systems of equations*. Cambridge University Press, Cambridge.
27. Ratschan, S. (2002). Approximate quantified constraint solving by cylindrical box decomposition. *Reliable Computing*, 8(1), 21–42.
28. Ratschan, S. (2006). Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4), 723–748.
29. Ratschan, S. (2008). Applications of quantified constraint solving over the reals bibliography. <http://www.cs.cas.cz/~ratschan/appqcs.html>.

30. Vu, X. H., Sam-Haroud, D., & Silaghi, M.-C. (2002). Approximation techniques for non-linear problems with continuum of solutions. In *Proceedings of the 5th international symposium on abstraction, reformulation and approximation, LNAI* (Vol. 2371, pp. 224–241). Springer-Verlag.
31. Vu, X.-H., Silaghi, M., Sam-Haroud, D., & Faltings, B. (2006). Branch-and-prune search strategies for numerical constraint solving. Technical Report LIA-REPORT-2006-007, Swiss Federal Institute of Technology (EPFL).
32. Zettler, M., & Garloff, J. (1998). Robustness analysis of polynomials with polynomial parameterdependency using bernstein expansion. *IEEE Transactions on Automatic Control*, 43(3), 425–431.