Aiding design with constraints: an extension of quad trees in order to deal with piecewise functions

Michel Aldanondo, Élise Vareilles, Khaled Hadj-Hamou, Paul Gaborit

Aiding design with constraints: an extension of quad trees in order to deal with piecewise functions

M. ALDANONDO*†, E. VAREILLES† K. HADJ-HAMOU‡ and P. GABORIT†

†Centre de Génie Industriel - Ecole des Mines d'Albi-Carmaux, Campus Jarlard - 81013 Albi CT Cedex 09 – France
‡Laboratoire GILCO - ENSGI – INP Grenoble, 46 Avenue Félix Viallet-38031 Grenoble, France

This paper deals with aiding preliminary design when considered as a constraint satisfaction problem (CSP). In this case, constraint filtering techniques provide some kind of interactive assistance to the designer. However, some kinds of numerical constraints corresponding with numerical relations cannot be filtered precisely with classical analytical filtering techniques such as interval arithmetic or box-consistency; it is therefore necessary to discretize them in order to include them in the CSP. To this end, quad trees (QT) have been proposed for binary constraints, or 2^k trees when more than two variables are considered; but QT assume that a constraint must be defined by a single numerical function. The aim of this paper is to show that QT techniques can be extended when a constraint is defined by a piecewise function or by a set of numerical functions defined on intervals. The first section recalls some basics relevant to the preliminary design problem and the interests of the CSP assistance. The second section presents the principles of the QT. The last section describes our contributions relevant to QT extensions dealing with piecewise functions.

Keywords: Preliminary design; Knowledge-based system; Constraint satisfaction problem; Quad trees

1. Introduction

The first sub-section recalls constraint satisfaction problem (CSP) basics and their usefulness in preliminary design. Then the industrial design problem, which is at the origin of our work, is briefly presented in the second sub-section. The third sub-section summarizes constraint types and filtering techniques and introduces the discretization need. The last sub-section introduces the quad tree (QT) idea and the aim of the paper.

1.1. Constraint satisfaction problems and preliminary design

Preliminary design defines the structure and main parameters of a product. Therefore, it manipulates partial description of the artefact, consisting of product parts (components) and properties (resistance, performance). In general, preliminary design is the stage at which the major design decisions are made: selection of the main components and valuation of the significant properties or parameters.

These decisions are made by using various sources of knowledge gathering field knowledge and past experiments. Unfortunately, during this stage, wrong decisions can be made by the designers. These errors can be very difficult to rectify later and can result in huge additional cost and time requirements. In order to avoid these errors and to help designers to make relevant decisions, the knowledge used in preliminary design can be extracted, validated and organized in knowledge-based systems

There are two main kinds of systems able to store knowledge and to use it to help decision making: those that

*Corresponding author. Email: aldanondo@enstimac.fr

gather all the previous experiments in a data base, and those that are based on an explicit reasoning model, gathering variables and relations. In such a reasoning model, the variables stand for the product parameters and its consisting parts, while the relations express the permissible combinations of values for theses parameters (field knowledge as: physical laws, manufacturing rules, design preferences, etc). A preliminary design task lends itself naturally to such formulation.

Such an explicit reasoning model can be easily associated to a constraint satisfaction problem. Indeed, a CSP is defined by a set of variables $\{Xi\}$, each of them defined on a finite domain $\{Di\}$, and a set of constraints $\{C\}$ that express the permissible combinations of values for the variables (Tsang 1993). Constraints can be either formalized as compatibility tables or mathematical expressions. In the case of preliminary design, the corresponding CSP restricts the space of feasible product solutions respecting the preliminary design requirements.

Interactive assistance during design is provided by filtering techniques that remove inconsistent values from the domain of the variables. Each time the designer inputs a value or a domain restriction on any product parameter, constraint filtering or propagation removes inconsistent values from the domain of definition of the remaining parameters.

The clear distinction between the problem definition and its resolution, the easily understood modelling concepts and the possibility of interactive resolution permit the consideration of the CSP framework as an appropriate tool for aiding preliminary design. Many studies have shown that this approach, embedded in a knowledge based system, could be a significant computer assistance for various design domains in the following examples: steel buildings (Gelle *et al*. 2000), automotive parts (Aldanondo *et al*. 2001), aeroplanes (Bensana *et al*. 2000), manufacturing operations (Geneste *et al*. 2000) or civil engineering (Lottaz *et al*. 1999).

However the feasibility and the usefulness of this approach are strongly dependent on the kinds of variables and the kinds of constraints which are necessary to formalize the knowledge relevant to the product that must be designed.

1.2. Design of heat treatment operation as a constraint satisfaction problem

One of the goals of the European project VHT (virtual heat treatment - project No G1RD-CT-2002-00835) was to design a knowledge-based system (KBS) in order to assist engineers in charge of the definition of heat treatment operations. The KBS relies on a case base reasoning system (CBR) and on a CSP-based processing system. For the CSP part, (i) some expert knowledge was collected and gathered in a CSP-based reasoning model and (ii) an interactive constraint propagation engine was designed and developed (Lamesle *et al*. 2005).

The delicate point of heat treatment operation design is to try to avoid distortions. According to heat treatment experts (Oliveira *et al*. 1986, David *et al*. 2003), distortions can result from any kind of choices relevant to the definition of the heat treatment operation. Consequently, the resulting constraints model gathers four sets of variables relevant to: (i) the geometry of the part, (ii) the material of the part, (iii) the resources required by the operation and (iv) the distortion characteristics. Constraints, linking these variables, correspond with compatibility tables, mathematical expressions but also with two-dimensional (2D) experimental charts or graphs as the one shown in figure 1(a) that shows some cooling curves with respect to phases transformation. Unfortunately, this kind of 2D charts cannot be approximated with a single mathematical expression, and are, mostly, fitted with a set of mathematical expressions defined on particular domains: three linear functions in the example of figure 1(a). This example comes from metallurgy, many others exist in engineering design as for example in figure 1(b) where the displacement of the extremity of a beam is considered with respect to the admissible load and relevant section shape.

1.3. Variables, constraints and filtering algorithms

When the variables of the problem are defined on symbolic domains, constraints can be represented by tables, gathering permissible combinations of values, and the corresponding CSP is discrete. In that case, many filtering algorithms, based on arc-consistency, have been proposed and a good survey can be found in Debruyne and Bessière (2001).

When the variables are defined on numerical domains, previous tables can be extended in order to express permissible combinations of intervals, and filtering algorithms can also be adapted in order to take into account these kinds of constraints in the CSP, as explained in Gelle (1998).

When the variables are numerical and each constraint is expressed by a mathematical expression (linear, nonlinear, equality or inequality), we can use two different filtering techniques: 2B-consistency proposed by Lhomme (1993) and box-consistency proposed by Benhamou *et al*. (1994). Both focus on optimizing the tightening of the feasibility space outer bounds: they approximate the effective solution space by a rough enclosing box (Sam 1995). Their principal difference lies in the fact that 2B-consistency requires the projection of the constraints on each of their variables while Box-consistency works directly on the original constraints by using interval Newton iterates.

In order to define a more precise and efficient representation of continuous solution space, Sam (1995) proposes the

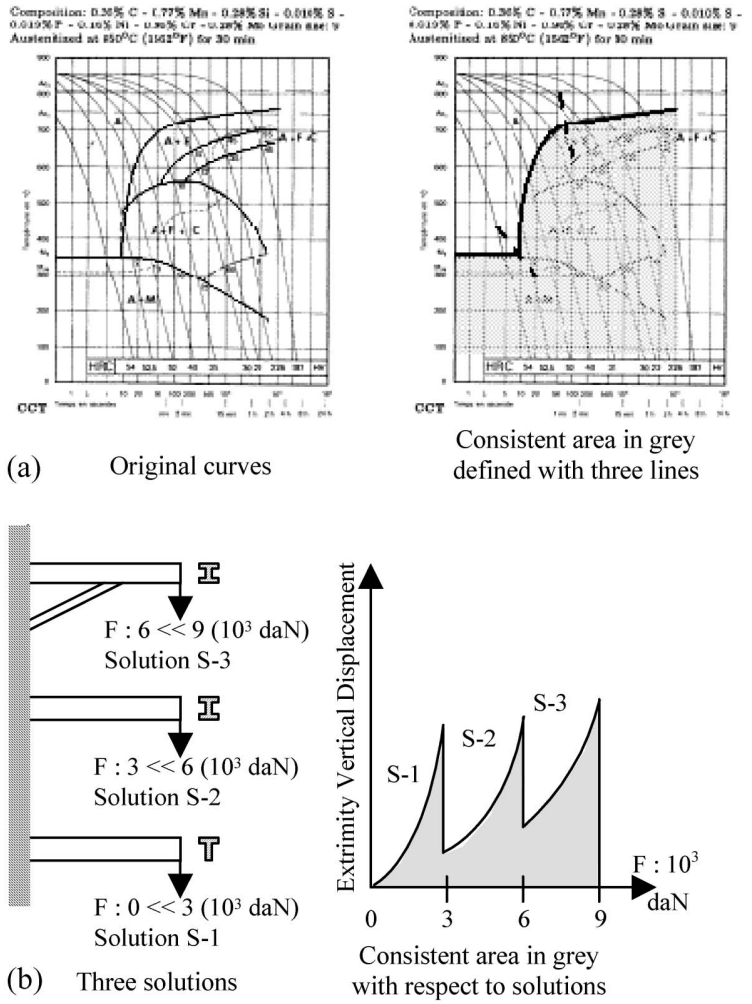

Figure 1. (a) Cooling curves and constraint as a piecewise function and (b) displacement data and constraint as a piecewise function.

use of QTs (Finkel and Bentley 1974) for binary constraints, or $2k$ trees when more than two variables are considered.

1.4. Quad trees, constraints as functions and piecewise functions

The main idea is to discretize the space defined by the variables belonging to each constraint. If the constraint acts on two variables (binary constraint), the 2D space is split into rectangles and provides a QT. When three variables are concerned, the space is decomposed into cubes and provides an OcTree. A constraint dealing with k variables will be decomposed into hyper-cubes and provide a 2^k tree.

We only consider, in this paper, binary constraints and relevant quad trees, for an easy to understand presentation but also because most of the experimental knowledge used in preliminary design to make a decision is represented as 2D charts or graphs in order to be manually used. Frequently, this kind of 2D graphs cannot be easily fitted with a single mathematical expression, and mostly, several mathematical expressions defined on different domains are necessary to approximate the graph with sufficient accuracy, as shown in figure 1.

Sam (1995) has shown that this approach was of interest in order to include constraints defined by various mathematical expressions in a constraint satisfaction problem. In order to be able, in preliminary design, to consider constraints defined by piecewise functions in the constraint model of the product, we propose to extend QT techniques to piecewise functions. The next section recalls QT basics and the last section shows how piecewise functions gathering equalities or inequalities, can be processed as QTs.

2. Quad trees: definition and generation

The aim of this section is to recall QT definition and to explain how QTs can be generated and integrated in a constraint satisfaction problem. Several examples illustrate the elements presented.

2.1. *Quad tree definition*

A QT is a hierarchical representation of a binary numerical constraint, $C(x,y)$, linking two variables, x and y, each one defined on a numerical domain, Dx and Dy. A QT gathers nodes defined as follows:

(a) each node n is defined by a pair of intervals (δx, δy),
(b) each node n is constrained by $C(x,$y),
(c) each node n has a colour: white, grey or black defined according to the consistency of the intervals (δx, δy) with the numerical constraint $C(x,y)$:
  (i) white means that all the pairs of values belonging to (δx, δy) are considered consistent with the constraint,
  (ii) black means that all the pairs of values belonging to (δx, δy) are considered not consistent with the constraint,
  (iii) grey means that the node n gathers both consistent and inconsistent pairs with respect to the constraint.
(d) each node n with a colour grey is split into four children: NW (north-west), SW (south-west), SE (south-east) and NE (north-east),
(e) two discretization steps, ε_x and ε_y, relevant to the two variables x and y, stop the Tree decomposing at a given accuracy level,
(f) when one of these two discretization steps is reached, the grey nodes are called *unitary* nodes and considered:
  (i) white or consistent with the constraint, if the goal is to forbid the rejection of pairs (x,y) which are consistent,
  (ii) black or inconsistent with the constraint, if the goal is to forbid the acceptation of pairs (x,y) which are inconsistent.
(g) the leaves of the resulting QT are therefore either black or white.

2.2. *Quad tree generation*

The generation of a QT can be launched at the initial search area defined by the domains of the variables x and y (Dx, Dy). The search area is decomposed recursively until the discretization steps (εx, εy) are reached. All kinds of binary numerical constraints can be represented by a QT: $f(x,$y$) \bullet 0$, with $\bullet$ belonging to $\{=, <, \leq, >, \geq\}$.

There are two ways to compute the colour of a node. The first one, proposed by Sam (1995), consists of using mathematical techniques to compute the intersections between the four sides of a node and the constraint. This computation can be a delicate problem according to the shape of the mathematical expression.

The second method, proposed by Lottaz (2000), consists of using interval arithmetic to verify if a node satisfies or not the constraint. Interval arithmetic (Moore 1966) extends real arithmetic to intervals by applying the operators of a formula to the endpoints of the intervals of its arguments. As interval arithmetic can over-estimate results, Lottaz (2000) has shown that it can happen that a node belonging completely to either a consistent or an inconsistent region has to be unnecessarily decomposed and explored.

In spite of this drawback, we have chosen this last approach to compute the colour of the nodes, mainly because it operates well whatever the shape of the mathematical expression is.

Each tree node is encoded using a succession of h digits, representing an integer in base $2h$, where the number of digits h corresponds to the height of the encoding node n in the Quad Tree. This encoding is based on a Peano's filled path with an N motif, arranged following Morton's order (Briggs and Peat 1991), as show in figure 2. This kind of encoding produces a unique code per node, corresponding to its geographic coordinates in base $2h$, as shown in figure 3.

2.3. *Quad tree example*

Figure 4 shows an example of a constraint $C(x,\ y)$: $y-x^3+0.1>0$ defined on numerical domains $Dx=[-2,$ 2] and $Dy=[-2,\ 2]$ with its relevant QT. The discretization steps are $\varepsilon x=1$ and $\varepsilon y=1$. Let us compute the consistency of the following nodes by using interval arithmetic:

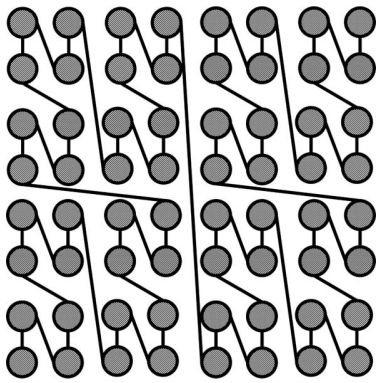

Figure 2. Peano's filled path with a N motif.

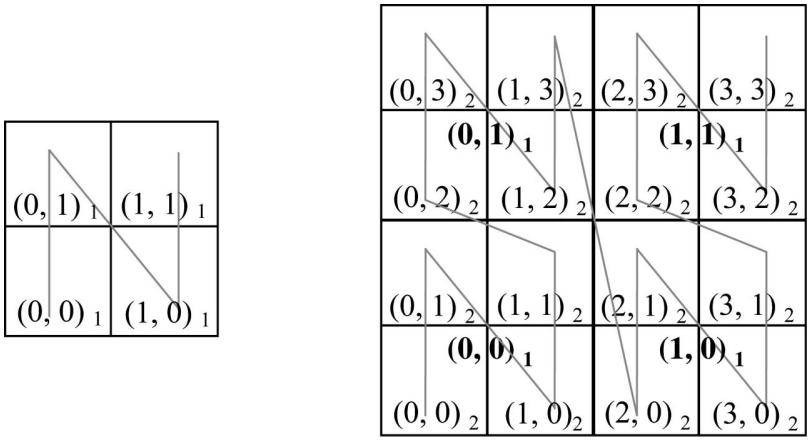

Figure 3. Encoding of the nodes in base 2^h.

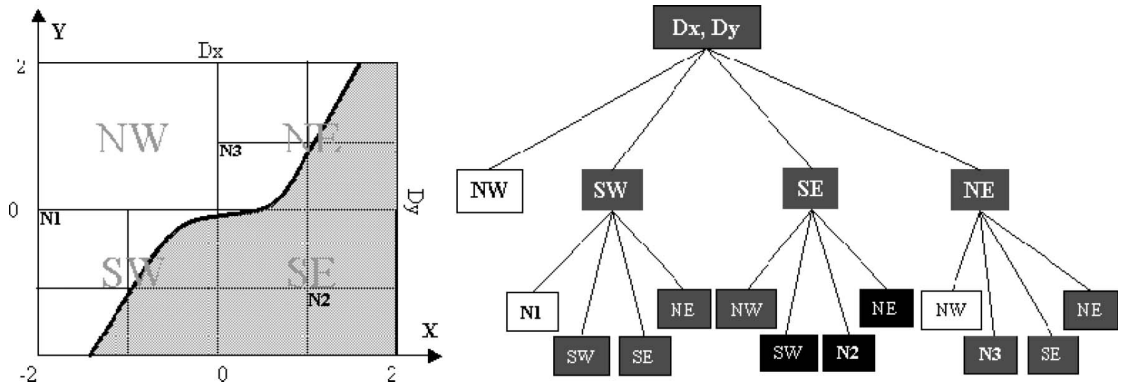

Figure 4. Numerical binary constraint and corresponding quad tree.

1. Let N1 be the node defined by a pair of intervals ($\delta x = [-2, -1]$ $\delta y = [-1, 0]$). N1 is white and completely consistent with the constraint because:

$$\begin{aligned} C([-2,-1],[-1,0]) &= [-1,0] - [-2,-1]^3 + [0.1,0.1] \\ &= [-1,0] - [-8,-1] + [0.1,0.1] \\ &= [0,8] + [0.1,0.1] \\ &= [0.1,8.1] > 0 \end{aligned}$$

true for all the interval

2. Let N2 be the node defined by a pair of intervals ($\delta x = [1,2]$ $\delta y = [-2, -1]$). N2 is black and completely inconsistent with the constraint because:

$$\begin{aligned} C([1,2],[-2,-1]) &= [-2,-1] - [1,2]^3 + [0.1,0.1] \\ &= [-2,-1] - [1,8] + [0.1,0.1] \\ &= [-10,-2] + [0.1,0.1] \\ &= [-9.9,-1.9] > 0 \end{aligned}$$

false for all the interval

3. Let N3 be the node defined by a pair of intervals ($\delta x = [0, 1]$ $\delta y = [0, 1]$). N3 is grey and partially consistent with the constraint because:

$$\begin{aligned} C([0,1],[0,1]) &= [0,1] - [0,1]^3 + [0.1,0.1] \\ &= [0,1] - [0,1] + [0.1,0.1] \\ &= [-1,1] + [0.1,0.1] = [-0.9,1.1] > 0 \end{aligned}$$

true and false on the interval

When the two discretization steps, εx and εy are reached, the grey leaves turn:

(a) white or consistent with the constraint, if the goal is to forbid the rejection of pairs (x, y) that are consistent, as shown in figure 5,
(b) black or inconsistent with the constraint, if the goal is to forbid the acceptance of pairs (x, y) that are inconsistent, as shown in figure 6.

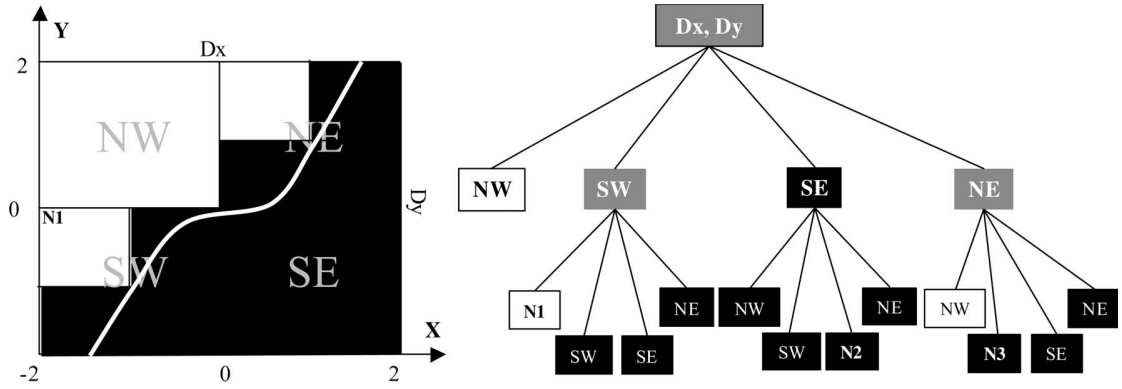

Figure 5. Final quad tree that forbids the rejection of consistent couples.

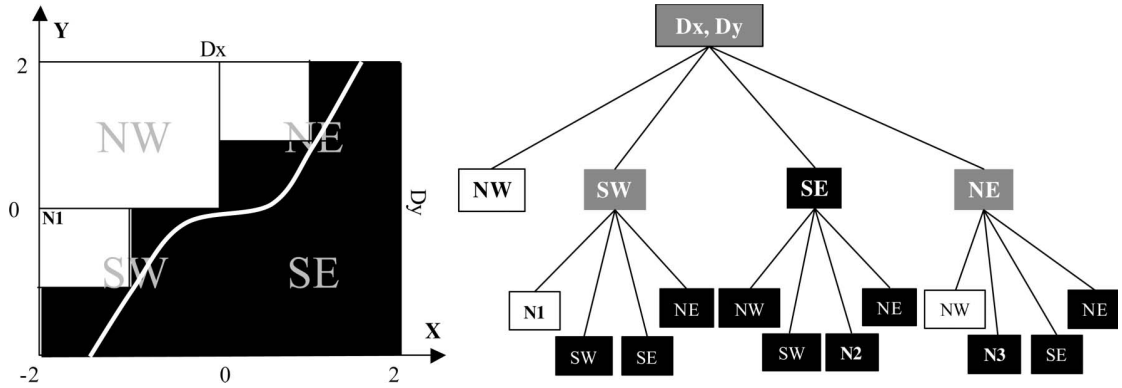

Figure 6. Final quad tree that forbids the acceptance of inconsistent couples.

2.4. Quad tree and constraint processing

In order to include QT in a CSP, the intervals (δx, δy) of the white nodes are gathered in a compatibility table that represents permissible combinations of intervals, as shown in the table 1 for the two previous QTs.

When more than one constraint acts on the same pair of variables, it is necessary to compute the intersection of the constraints and therefore the intersection of their QT. Sam (1995) proposed calling this operation 'fusion' and denoting it $\oplus$. The fusion is simply defined by the occlusion of consistent regions by inconsistent ones. The fusion of two QTs is therefore implemented by combining the colours of each pair of corresponding nodes (i.e. nodes covering the same region in the search area). The colours of the nodes are in a certain order and it is assumed that: white < grey < black. The rule for determining consistency of a node n obtained by fusion of two nodes $n1$ and $n2$ belonging to two QTs, is expressed as:

$$\begin{aligned}\text{colour}(n) &= \text{colour}(n1 \oplus n2)\\ &= \max(\text{colour}(n1), \text{colour}(n2))\end{aligned}$$

Table 1. Compatibility table.

X	Y
Compatibility table of figure 5	
[−2, 0]	[0, 2]
[−2, 0]	[−2, 0]
[0, 1]	[−1, 0]
[0, 2]	[0, 2]
Compatibility table of figure 6	
[−2, 0]	[0, 2]
[−2, −1]	[−1, 0]
[0, 1]	[1, 2]

This operation is illustrated in figure 7, where two constraints C1: $f1(x,y) > 0$ and C2: $f2(x,y) < 0$, with hatched inconsistent areas, are intersected.

3. Extension of quad trees

The aim of this section is to extend the QT approach in order to be able to take into account constraints defined

by piecewise functions. A first sub-section defines what we mean by piecewise functions and introduces what we call information grades. Following sub-sections present QTs which deal with constraint gathering equalities, $f(x,y)=0$, and inequalities, $f(x,y)>$ or <0.

3.1. *Piecewise functions and information grades*

3.1.1. Piecewise functions. A piecewise function $C(x, y)$ gathers a set of mathematical expressions, called pieces and noted $fi(x,y)$. A piece $fi(x,y)$ is defined on a domain (Dxi,Dyi) and is either an equality constraint $fi(x, y)=0$ or an inequality constraint $fi(x,y) \bullet 0$, with $\bullet$ belongs to $\{<, \leq, >, \geq\}$. A piecewise function $C(x, y)$ is either a set of equality constraints or a set of inequality constraints.

We notice that a region (dx,dy) of the search area (Dx,Dy), can be covered by more than the domain of definition of one piece (Dxi,Dyi) or by none at all.

Figure 8 shows two examples of piecewise constraints: on the left side, the constraint named 'Wave', gathers fives pieces, whereas on the right side, the constraint, named 'Potato', gathers four. Each grey rectangle corresponds with the domain of definition of each piece $fi(x,y)$. We can see that:

(a) for both constraints, some parts of the search area are not covered by any definition domains (Dxi,Dyi),
(b) for the constraint named 'Potato', a region is covered by two domains of definition, the first one corresponding to $f1(x,y)$ and the second one to $f3(x,y)$.

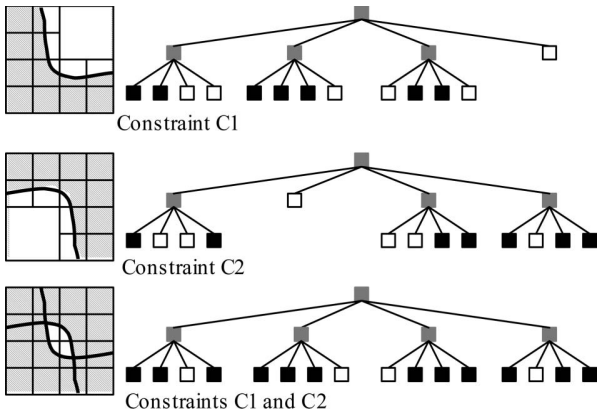

Figure 7. Fusion of quad trees.

Some hypotheses on the general outline of piecewise constraints are necessary to guarantee the existence of a border between the consistent and inconsistent regions:

(a) the general outline of the piecewise constraint $C(x,y)$ must be closed in case of inequality constraints, and must be bounded in case of equality ones,
(b) two pieces of an inequality constraint should not cross each other (the border between the consistent and inconsistent regions must be clearly defined),
(c) in case of inequality constraints, all the pieces must be consistent with the others in order to define correctly consistent and inconsistent regions.

All the hypotheses are considered verified when the QT generation is launched.

3.1.2. Grades of information. As a node n of a QT can either intersect several domains of definition of the pieces (Dxi,Dyi) or none at all, we need to characterize different types of nodes with what we call a grade of information. We propose four grades of information according to the two following types of intersections:

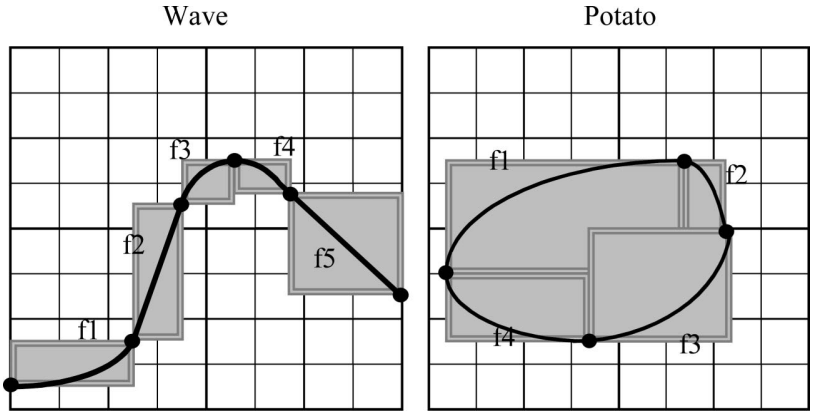

Figure 8. Examples of piecewise constraint.

(a) between a node and the domain of definition of a piece (Dxi,Dyi),
(b) between a node and a piece itself $fi(x,y)$.

These intersections are computed by using interval analysis. Four types of node are then defined:

(1) empty nodes which do not have any information to determine their consistency with the piecewise constraint. A node n defined on (δx, δy) is an empty node if it does not intersect any domain of definition of the pieces (Dxi,Dyi):

$$(\delta x, \delta y) \cap (Dxi, Dyi) = \emptyset \quad \text{for the definition domain of any piece}$$
$$(\delta x, \delta y) \cap fi(x, y) = \emptyset, \quad \text{for any piece}$$

(2) poorly informed nodes which don't have enough information to determine their consistency with the piecewise constraint. A node n defined on (δx, δy) is a poorly-informed node if (i) it intersects partially or completely, at least one domain of definition of a piece (Dxi,Dyi) (ii) but does not intersect any piece $fi(x,y)$.

$$(\delta x, \delta y) \cap (Dxi, Dyi) \neq \emptyset, \quad \text{at least for the definition domain of one piece}$$
$$(\delta x, \delta y) \cap fi(x, y) = \emptyset, \quad \text{for any piece}$$

(3) informed nodes which have enough information to determine their consistency with the piecewise constraint. A node n is an informed node if it intersects one and only one piece $fi(x,y)$.

$$(\delta x, \delta y) \cap (Dxi, Dyi) \neq \emptyset, \quad \text{at least for the definition domain of one piece}$$
$$(\delta x, \delta y) \cap fi(x, y) \neq \emptyset, \quad \text{for one and only one piece}$$

(4) over-loaded nodes which have too much information to determine their consistency with the piecewise constraint. A node n is an over-loaded node if it intersects more than one piece $fi(x,y)$.

$$(\delta x, \delta y) \cap (Dxi, Dyi) \neq \emptyset, \quad \text{at least for the definition domain of one piece}$$
$$(\delta x, \delta y) \cap fi(x, y) \neq \emptyset, \quad \text{for more than one piece}$$

Empty and poorly-informed nodes are called ignorant nodes. The solution space can be therefore divided into six grades of information: consistent, empty, poorly informed, informed, over-loaded and inconsistent.

If we have a look at the previous examples, 'Wave' and 'Potato', we can see, on figure 9, that the solution space contains the four information grades defined above: (E) for empty nodes, (P) for poorly-informed nodes, (I) for informed nodes and (O) for over-loaded nodes.

3.2. Generation of quad trees gathering a set of equality constraints

The generation of QT gathering a set of equality constraints is recursive, based on the grades of information and follows the ideas of the classical QT generation:

(a) each node n is defined by a pair of intervals (δx, δy),
(b) if a node n is an ignorant one, both empty or poorly-informed ones, it is coloured black,
(c) if a node n is an over-loaded one (more than one piece intersect the node), it is coloured grey. Each grey node is split into four children: NW (north-west), SW (south-west), SE (south-east) and NE (north-east), whose information grade (empty, poorly informed, informed or over-loaded informed) has to be computed,

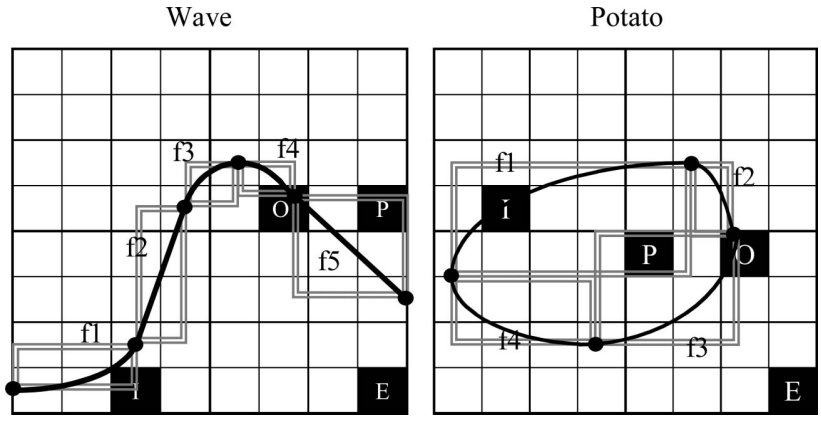

Figure 9. Information grades.

(d) if a node is an informed one (only one piece $fi(x,y)$ intersects the node), classical Quad Tree generation is launched from the isolated piece $fi(x,y)$ (cf.sub-section 2.1). Consistency is computed and colours (white or black) are given to the resulting nodes,
(e) two discretization steps, εx and εy, relevant to the two variables x and y, allow the Tree decomposition to stop at a given accuracy level. When one of these two discretization steps is reached, the grey unitary node becomes white or consistent with the constraint because we are dealing with equalities.

In order to illustrate the previous propositions, let us consider the constraint named 'Wave' of figure 8, and its Quad Tree represented in figure 10, on which we note E for empty nodes and P for poorly-informed ones.

Let us look in detail at the decomposition of the upper left node of figure 10, encoded $(0,1)_1$ with Peano's encoding. This node is an over-loaded one, containing two pieces: f2 and f3. It is coloured in grey and split into four child nodes:

(a) its first child, encoded $(0, 3)_2$, is an empty node: its colour is therefore black,
(b) its second child, encoded $(0, 2)_2$, is also an empty node: its colour is therefore black,
(c) its third child, encoded $(1, 2)_2$, is an over-loaded node, containing f2 and f3: its colour is therefore grey,
(d) and its fourth child, encoded $(1, 3)_2$, is an empty node: its colour is therefore black.

Now, let us consider its child node encoded $(1, 2)_2$. This node is split once more into four child nodes because the precision has not been reached:

(a) its first child, encoded $(2, 5)_3$, is a empty node: its colour is therefore black,
(b) its second child, encoded $(2, 4)_3$, is a poorly-informed node which intersects the domain of definition of f2: its colour is therefore black,
(c) its third child, encoded $(3, 4)_3$, is a unitary over-loaded node, containing two pieces f2 and f3: its colour is therefore white,
(d) and its fourth child, encoded $(3, 5)_3$, is a unitary informed node, intersecting only the piece f3: its colour is therefore white.

3.3. Generation of quad trees gathering a set of inequality constraints

The idea of the previous section, associating a grade of information to each node, is kept, but the generation of the QT is a little bit more complex because ignorant nodes can belong either to consistent or inconsistent regions. Therefore, the generation is achieved in two steps:

(1) First, during the recursive decomposition, we need to mark the information grade of each node with a specific colour,
(2) Second, when the decomposition stops, the consistency of each ignorant node is found, thanks to the propagation of the consistent and inconsistent regions.

The first step of the QT generation follows the principles below:

(a) a node n is defined by a pair of intervals $(\delta x, \delta y)$,
(b) if a node n is an ignorant one, both empty or poorly-informed ones, it is coloured red,
(c) if a node n is an over-loaded one (more than one pieces intersects the node), it is coloured:
  (i) orange if it is *unitary* (the accuracy level, εx and εy, is reached),
  (ii) grey otherwise and split into four children: NW (north-west), SW (south-west), SE (south-east) and NE (north-east), whose grade

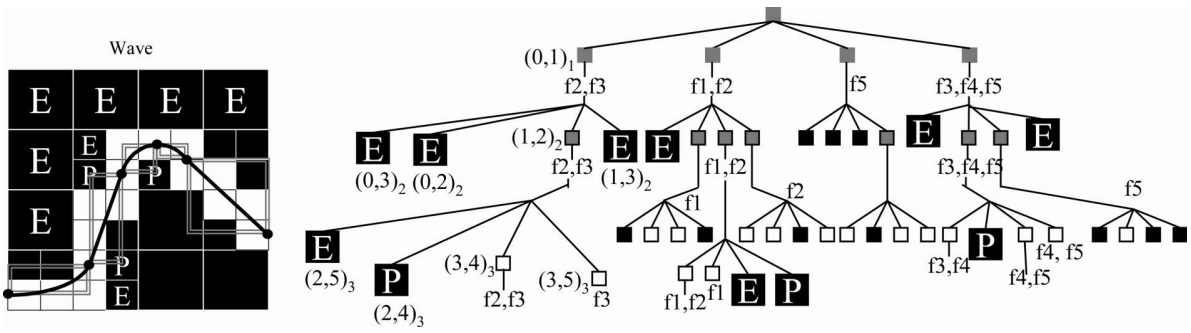

Figure 10. Resulting Quad Tree.

of information (empty, poorly-informed, informed or over-loaded informed) has to be computed and marked,

(d) if a node n is an informed one (only one piece $fi(x,y)$ intersects the node), it is coloured:
  (i) yellow if it is unitary (the accuracy level, εx εy, is reached),
  (ii) grey otherwise and classical QT generation is launched from the isolated piece (cf. subsection 2.1). Consistency is computed and colours (white or black) are given to the resulting nodes.

(e) two discretization steps, εx and εy, relevant to the two variables x and y, allow the Tree decomposition to stop at a given accuracy level. When one of these two discretization steps is reached, the second step of the generation is launched.

At the end of the first step, the leaves of the QT can be coloured either in:

(a) white for consistent nodes,
(b) black for inconsistent nodes,
(c) yellow for unitary informed nodes,
(d) orange for unitary over-loaded nodes
(e) or red for ignorant nodes, the empty and the poorly-informed ones.

In order to illustrate these different colours, let us consider the outside area and the border of the constraint named 'Potato' of figure 8, as the consistent region. When the first step is finished as shown in figure 11, the QT is multi-coloured with red, yellow and orange leaves. We note on figure 11, 'R' for red nodes, 'Y' for yellow nodes and 'O' for orange nodes.

Let us look in detail at the decomposition of the lower left node of figure 11, encoded $(0,0)_1$ with Peano's encoding. This node is an over-loaded one, containing three pieces: f1, f3 and f4. It is coloured in grey and split into four child nodes:

(a) its first child, encoded $(1, 0)_2$, is an over-loaded node, containing f1 and f4: its colour is therefore grey,
(b) its second child, encoded $(0, 0)_2$, is an informed node, containing only f4, the classical Quad Tree generation is launched from the isolated piece f4, and permit us to conclude that the whole node is consistent with the piecewise constraint and therefore white,
(c) its third child, encoded $(0, 1)_2$, is an over-loaded node, containing f3 and f4: its colour is therefore grey,
(d) and its fourth child, encoded $(1, 1)_2$, is an poorly-informed node, intersecting only the domain of definition of f4: its colour is therefore red.

Now, let us consider its child node encoded $(1, 0)_2$. This node is split once more into four child nodes because the precision has not been reached:

(a) its first child, encoded $(0, 3)_3$, is a unitary over-loaded node, containing f1 and f4: its colour is therefore orange,
(b) its second child, encoded $(0, 2)_3$, is also an unitary over-loaded node, containing f1 and f4: its colour is therefore orange,
(c) its third child, encoded $(1, 2)_3$, is a unitary informed node, containing only f4: its colour is therefore yellow,
(d) and its fourth child, encoded $(1, 3)_3$, is an poorly-informed node, intersecting the domains of definition of f1 and f4: its colour is therefore red.

The second step of this QT generation consists of propagating the consistent and inconsistent regions from the nodes which know their consistency (the yellow, black and white nodes), to those which are ignorant (the red ones). In order to propagate the consistent and the inconsistent regions, the neighbours of the yellow, black and white nodes must be identified. The selection of the relevant neighbours of a node n is made thanks to the

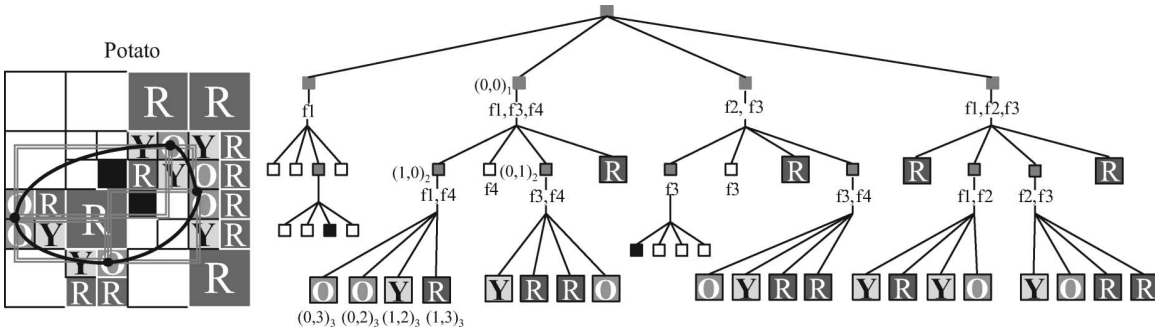

Figure 11. Quad tree at the end of the first step.

unique encoding per node. We notice that a node n can have neighbours with a higher or a lower height than its own: the encoding of its neighbours can have a different number of digits than it owns.

The propagation of the consistent and inconsistent regions is done in four steps:

(a) first, yellow nodes tell their red neighbours on which side of the border they belong to, by using interval arithmetic. If a red neighbour belongs to the consistent side, it turns white, otherwise, black.
(b) then, black nodes indicate to their red neighbours that they belong to the inconsistent region and therefore the red neighbours turn black.
(c) thirdly, white nodes indicate to their red neighbours that they belong to the consistent region. The red neighbours turn white.
(d) finally, yellow and orange nodes turn:
    (i) white or consistent with the piecewise constraint, if the goal is to forbid the rejection of pairs (x, y) that are consistent,
    (ii) black or inconsistent with the piecewise constraint, if the goal is to forbid the acceptation of pairs (x, y) that are inconsistent.

In our 'Potato' example, figure 12 illustrates this first propagation step, with the small arrows showing how previous red nodes become white. Then, as there are no contiguous red and black nodes, there is no need to propagate inconsistent region, so the second step is not necessary. The third step leads to figure 13 where the red neighbours turn white as indicated by the small arrows. Finally in figure 14, yellow and orange nodes turn white because the border is kept in the consistent region. At the end of the second step, all the leaves of the Quad Tree are either white or black and the consistent and inconsistent regions are identified.

3.4. *Extended quad trees and constraint processing*

The proposed, extended QTs, can be intersected without any problem with original QTs. Their integration in a CSP can be achieved thanks to compatibility tables gathering permissible combinations of intervals, as stated in subsection 2.3.

The proposed, extended QTs can also sometimes avoid the need to compute 'fusions' of QTs. As the proposed QT is defined for a set of mathematical expressions, it is possible to provide the fusion of two QTs in a single QT generation. If we consider the examples at figure 15, where the inconsistent values are hatched, it is clear that for each example a single QT can be generated, instead of a process including two independent generations and a fusion operation.

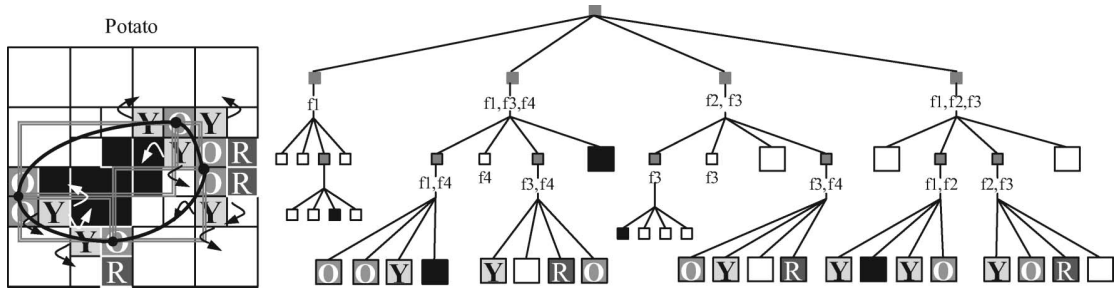

Figure 12. Propagation from yellow nodes to red ones.

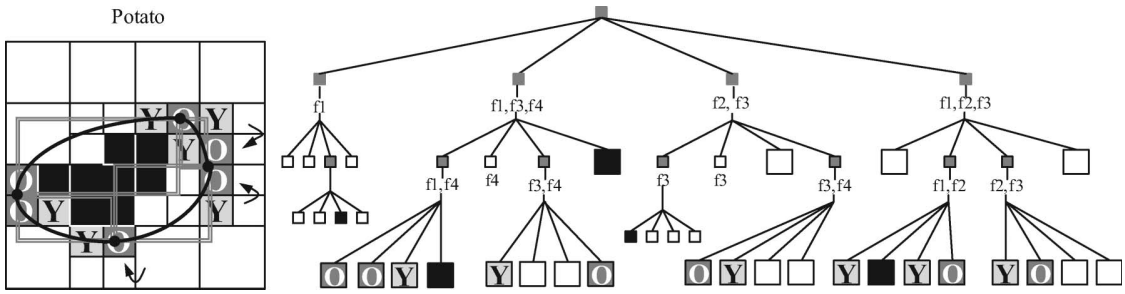

Figure 13. Propagation from white nodes to red ones.

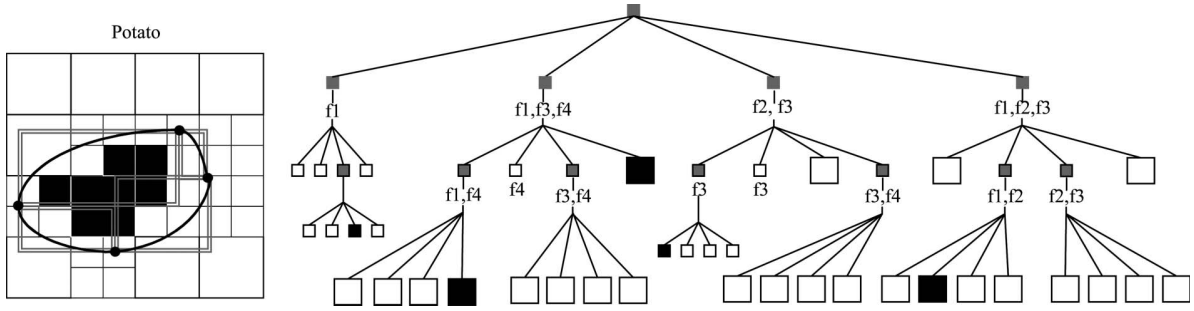

Figure 14. Resulting quad tree.

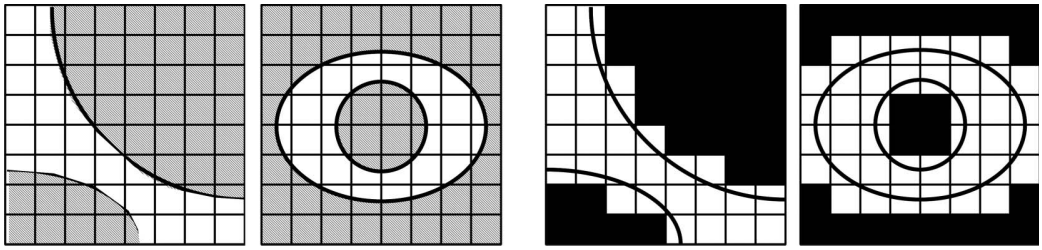

Figure 15. Correct quad tree corresponding with the fusion of two quad trees.

4. Conclusions

The aim of this paper has been to propose an approach that permits the handling of constraints, given as graphs or charts, in a constraint satisfaction problem. This sort of problem is frequently encountered in engineering, and especially in preliminary design where knowledge comes, most of the time, from experience. Frequently, such constraints cannot be easily approximated with a single mathematical expression, and are, mostly, fitted with a set of mathematical expressions defined on particular domains.

QTs, proposed by Sam (1995), allow single mathematical constraint to be taken into account in constraints satisfaction problems: the search area is split recursively into rectangles defining nodes, until a certain accuracy level is reached. During the decomposition, the consistency of each node is computed by using intervals arithmetic and is marked with a specific colour. We have extended this method to piecewise constraints, with respect to some insubstantial assumptions on their general outline.

Our method follows the idea of the classical QTs (recursive decomposition) but the generation of QTs of piecewise constraints is somewhat more complex. First, we have to identify the grade of information of each node. Indeed, some regions can be covered by several domains of definition of the pieces or by none at all. This kind of area cannot alone determine their consistency with the piecewise constraint. Four grades of information have been identified to characterize the different types of nodes. When the grade of information of each node is found, the consistent and inconsistent regions are propagated from the nodes which know their consistency to those which do not. The consistent and inconsistent regions defined by the piecewise constraint are then established.

The mechanism of 'fusion' proposed by Sam (1995), to compute the intersection of constraints represented by QTs, can be applied to QTs associated to piecewise constraints, without any particular problems.

This study has been necessary when designing a knowledge based system in order to take into account experimental knowledge, relevant to the heat treatment domain during a European project (project No G1RD-CT-2002-00835). The detailed algorithms relevant to Quad Tree generation and integration in a constraint satisfaction problem can be found in Vareilles (2005).

References

Aldanondo, M., Lamothe, J. and Hadj-Hamou, K., Configurator and CAD Modeler: gathering the best of two worlds. *Configuration Workshop of the 17th International Joint Conference on Artificial Intelligence*, Seattle, USA, 2001, pp. 1–7.

Benhamou, F., McAllester, D. and van Hentenryck, P., CLP(intervals) revisited. *Proceedings of the International Logic Programming Symposium 94,* Ithaca, NY, USA, 1994.

Bensana, E. and Mulyanto, T., A generic approach for conceptual design based on object oriented and constraint logic programming, *4th International Conference on Engineering Design and Automation*, Orlando, USA, 2000.

Briggs, J. and Peat, D., *Turbulent mirror: an illustrated guide to chaos theory and the science of wholeness*, 1989 (Harper & Row: New York).

David, P., Veaux, M., Vareilles, E. and Maury, J., Virtual heat treatment tool for monitoring and optimising heat treatment process. *2nd International Conference on Thermal Process Modelling and Computer Simulation*, Nancy, 2003.

Debruyne, R. and Bessière, C., Domain filtering consistencies. *J. Artific. Intell. Res.*, 2001, **14**, 205–230.

Finkel, R. and Bentley, J.L., Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 1974, **4**, 1–9.

Gelle, E., On the generation of locally consistent solution spaces in mixed dynamic constraint problems, PhD thesis EPFL, Lausanne, Switzerland, 1998.

Gelle, E., Faltings, B. and Smith, I., Structural engineering design support by constraint atisfaction. *Artificial Intelligence in Design'00*, 2000, pp. 311–331 (Kluwer: Dordrecht).

Geneste, L., Ruet, M. and Monteiro, T., Configuration of a machining operation. *ECAI Workshop on Configuration*, Berlin, Germany, 2000, pp. 44–49.

Lamesle, P., Vareilles, E. and Aldanondo, M., Towards a knowledge based system (kbs) for a qualitative distortion prediction for heat treatments. *First International Conference on Distortion Engineering (IDE)* Bremen Germany, 2005.

Lhomme, O., Consistency techniques for numerical CSPs. *IJCAI*, 1993.

Lottaz, C., Collaborative design using solution spaces. PhD Thesis EPFL, Lausanne, Switzerland, 2000.

Lottaz, C., Clement, D., Faltings, B. and Smith, I., Constraint based support for collaboration in design and construction. *J. Comput. Civil Engng*, 1999, **13**, 23–35.

Moore, R.E., *Intervals analysis*, 1996 (Prentice Hall: Englewood Cliffs, NJ).

Oliveira, M., Moreira, A., Loureiro, A., Denis, S. and Simon, S., Effet du mode de refroidissement sur les déformations produites par la trempe martensitique, *Actes du 5e congrès international sur les traitements thermiques des matériaux*, Budapest, 1986.

Sam, D., Constraint consistency techniques for continuous domains, PhD Thesis EPFL, Lausanne, Switzerland, 1995.

Tsang, E., *Foundations of Constraints Satisfaction*, 1993 (Academic Press: London).

Vareilles, E., Conception et approches par propagation de contraintes: contribution à la mise en oeuvre d'un outil d'aide interactif. PhD Thesis, Ecole des Mines d'Albi-Carmaux, France, 2005.