How to take into account piecewise constraints in constraint satisfaction problems

Élise Vareilles, Michel Aldanondo, Paul Gaborit

How to take into account piecewise constraints in constraint satisfaction problems

Élise Vareilles *, Michel Aldanondo, Paul Gaborit

École des mines d'Albi Centre de Génie Industriel, Université de Toulouse, Route de Teillet Campus Jarlard, 81013 Albi Cedex 09, France

A B S T R A C T

A particular data structure named a Quad Tree allows a better representation of solution space of binary continuous constraints $\boldsymbol{C}(x_1, x_2)$, than classical continuous consistencies. The generation and integration of this data structure do not raise any particular problem for continuous constraints defined by only one mathematical formula [Sam, D., 1995. Constraint consistency techniques for continuous domains. Ph.D. Thesis, École Polytechnique Fédérale de Lausanne]. In this paper, we propose to extend the method of generating Quad Trees in order to take into account, in *CSPs*, binary continuous constraints defined by a piecewise constraint, i.e. a set of functions defined on intervals. The first section presents the industrial requirements which led us to take into account this type of constraint in *CSP*. The second section recalls the principles of the Quad Tree. The last section describes our contributions relevant to Quad Tree extensions dealing with piecewise constraints.

Keywords:
Design
Piecewise constraints
Constraints satisfaction problem
Quad Tree
Filtering

1. Introduction

A particular data structure named a 2^k Tree (Sam, 1995) allows a better representation of continuous solution space of continuous constraints as defined by Definition 1, than continuous consistencies, such as 2B-consistency (Lhomme, 1993) and Box-consistency (Benhamou et al., 1994).

Definition 1. A continuous constraint $\boldsymbol{C}(x_1, \ldots, x_k)$ is an arbitrary equality or inequality involving variables $x_1, \ldots, x_k$ ($x_1, \ldots, x_k \in D_{x_1}, \ldots, D_{x_k}$ and $D_{x_1}, \ldots, D_{x_k} \in \mathscr{R}$).

Indeed, both consistencies 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) has proposed the use of Quad Trees (Samet, 1984) for binary continuous constraints and of 2^k-Trees for constraints involving k variables. The search area of a constraint is split recursively into rectangles or hypercubes until a certain accuracy level is reached. During recursive decomposition, the consistency with the constraint of each node is computed and marked with a particular colour. The integration of this data structure in CSP does not raise any particular problem for continuous constraints defined by only one continuous function (Sam, 1995). In this paper, we propose to extend the method of generating Quad Trees in order to take into account in *CSPs*, binary continuous constraints defined by a piecewise constraint (a set of functions defined on intervals). Our method, with respect to some insubstantial assumptions, follows the idea of classical Quad Tree (recursive decomposition), but the generation of Quad Trees of piecewise constraints is somewhat more complex.

The first section presents the industrial requirements which led us to take into account such binary constraints in *CSP*. The second section recalls what a Quad Tree is and how it is generated from a binary continuous constraint. The third section defines what piecewise constraints are and what the different grades of information we need to generate relevant Quad Trees. Then, the generation of Quad Trees of piecewise constraints is presented for constraints assembling inequalities and equalities.

2. Industrial requirements of piecewise constraints

This study has been necessary when designing a knowledge based system in order to take into account experimental knowledge, relevant to heat treatment domain during the European project *VHT* (Virtual Heat Treatment—Project no. G1RD-CT-2002-00835). Some expert knowledge has been collected and assembled in a CSP based reasoning model and an interactive constraint propagation engine has been designed and developed (Aldanondo et al., 2005).

* Corresponding author. Tel.: +33 563 493 092; fax: +33 563 493 183.
E-mail address: vareille@enstimac.fr (É. Vareilles).

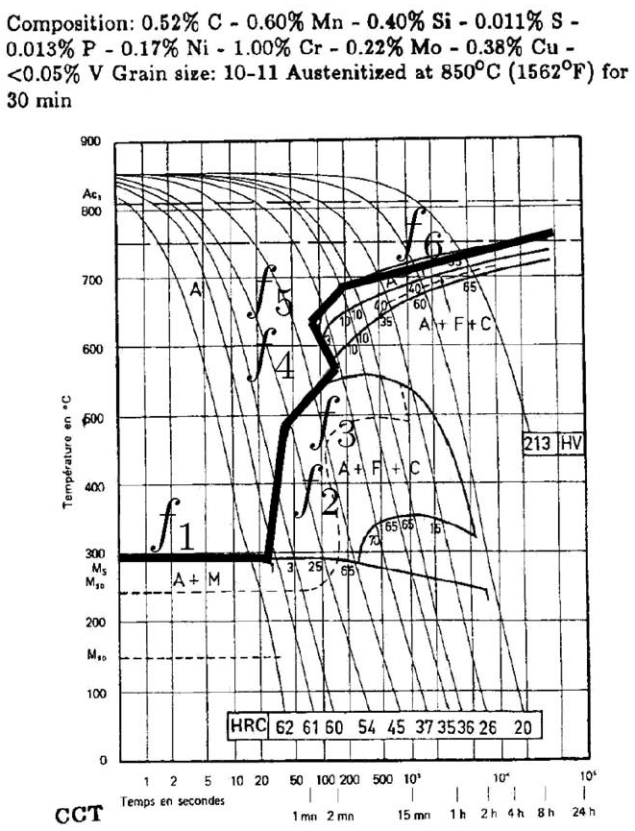

Fig. 1. Cooling curves and piecewise constraint.

The critical point of heat treatment operation design is trying to avoid distortions. According to heat treatment experts (David et al., 2003), distortions can result from any kind of choice relevant to the definition of the heat treatment operation. Consequently, the resulting constraints model is composed of 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 (Vareilles et al., 2007). The constraints linking these variables correspond with compatibility tables and mathematical expressions but also with 2D experimental graphs such as the one shown in Fig. 1 that shows some cooling curves with respect to transformating phases. The latest model is presented in Lamesle et al. (2005).

Unfortunately, experimental graphs cannot be approximated with a single mathematical expression, and are, most often, fitted with a set of mathematical expressions defined on particular domains: six linear functions in the example of Fig. 1. This example comes from metallurgy. Many others exist in engineering design as for example, the models in fluid mechanics involving the Reynolds dimensionless number whose value points to different types of ways the fluids flow (laminar, transient, turbulent) corresponding to fluid mechanics laws (Chenouard et al., 2007) or the displacement of the extremity of a beam with respect to the admissible load and relevant section shape, as shown in Fig. 2.

3. Quad Tree: definition and generation

The aim of this section is to recall Quad Tree definition and to explain how Quad Trees can be generated and integrated in a *CSP*. Several examples illustrate the elements presented.

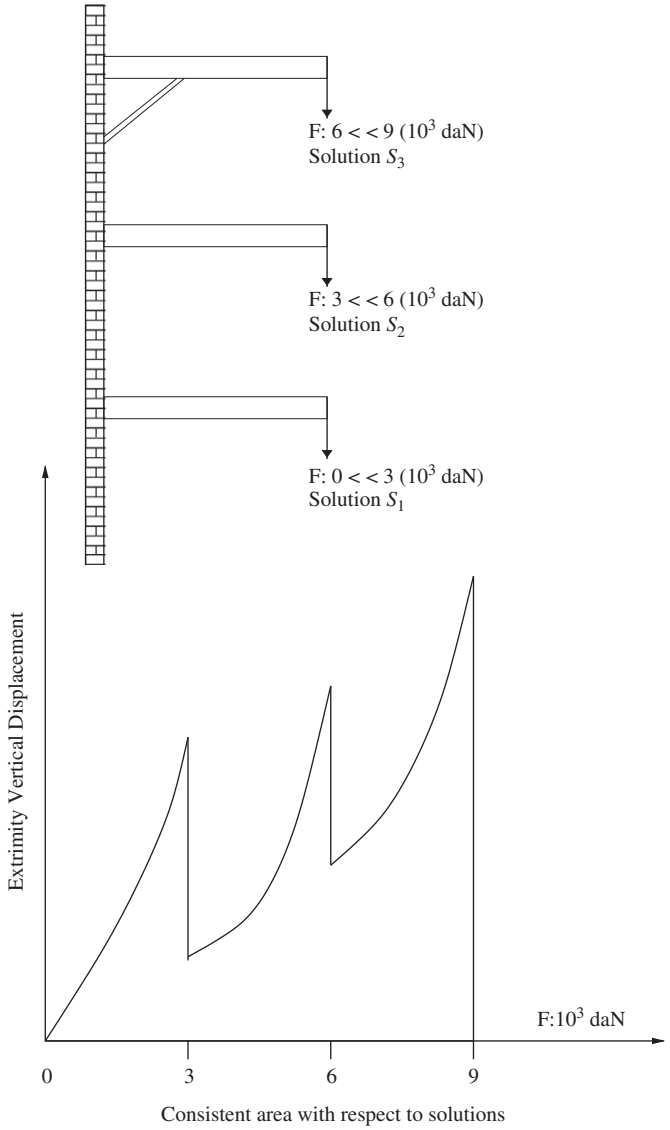

Fig. 2. Displacement data and piecewise constraint.

3.1. Quad Tree definition

Definition 2. A Quad Tree is made up of nodes defined as followed:

- each node n is defined by a pair of intervals (d_n^x, d_n^y),
- each node n is constrained by $\boldsymbol{C}(x, y)$ which is a constraint as defined by 1,
- each node n has a unique code, corresponding to its coordinates,
- each node n has a colour: `white`, `grey` or `blue` defined according to the consistency of the intervals (d_n^x, d_n^y) with the continuous constraint $\boldsymbol{C}(x, y)$,
  - `white`: if the region (d_n^x, d_n^y) is consistent with the constraint $\boldsymbol{C}(x, y)$,
  - `blue`: if the region (d_n^x, d_n^y) is not consistent with the constraint $\boldsymbol{C}(x, y)$,
  - `grey`: if the region (d_n^x, d_n^y) is both consistent and inconsistent with the constraint $\boldsymbol{C}(x, y)$.
- each `grey` node is split into four children notated *north-west* (NW), *south-west* (SW), *south-east* (SE) and *north-east* (NE),

each of whose consistency with the constraint $\boldsymbol{C}(x,y)$ has to be checked and marked by a colour,
- two discretization steps ε_x and ε_y relevant to the two variables x and y, stop the Quad Tree decomposition at a given accuracy,
- when one of these discretization steps is reached, the discretization stops. The grey nodes are then called *unitary* nodes and are considered:
  - consistent with the constraint $\boldsymbol{C}(x,y)$ if only the regions which are completely inconsistent need to be excluded: they turn `white`,
  - inconsistent with the constraint $\boldsymbol{C}(x,y)$ if only the regions which are completely legal with the constraint $\boldsymbol{C}(x,y)$ need to be kept: they turn `blue`.

The leaves of the resulting Quad Tree are therefore either `blue` or `white`.

Each node n is encoded using a succession of h digits, representing an integer in base 2^h, 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 shown in Fig. 3. This kind of encoding produces a code unique to each node, corresponding to its geographic coordinates in base 2^h.

3.2. Quad Tree generation

The generation of a Quad Tree can be launched at the initial search area defined by the domains of the variables x and y (D_x, D_y). The search area is decomposed recursively until either of the discretization steps, ε_x and ε_y, is reached. All kinds of binary continuous constraints, Definition 1, can be represented by a Quad Tree.

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

The other one, proposed by Lottaz (2000), consists in using interval arithmetic to verify if a node n satisfies the constraint or not. 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. If a node n completely satisfies the constraint, it turns `white`. If it partially satisfies the constraint, it turns `grey`. In the remaining case, the node is inconsistent with the constraint and turns `blue`. As interval

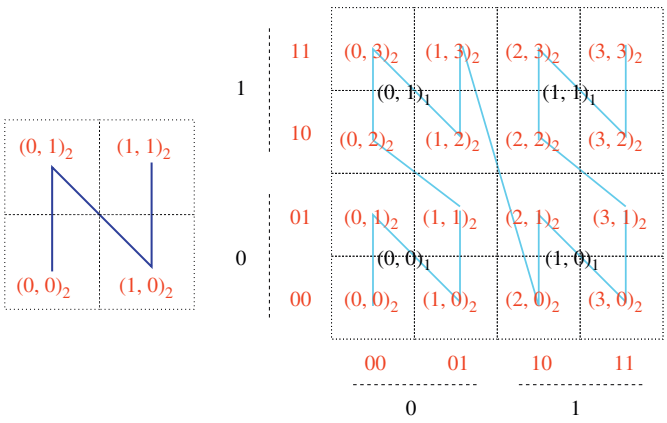

Fig. 3. Peano's filled path with an N motif, arranged following Morton's order.

arithmetic can over-estimate results, Lottaz (2000) has shown that it can happen that a node n belonging completely to either a consistent or an inconsistent region has to be decomposed and explored unnecessarily. 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.

3.3. Quad Tree example

For instance, Fig. 4 shows the Quad Tree corresponding to the constraint $\boldsymbol{C} : y - x^3 \geqslant 0$ where $\varepsilon_x = 0.0625$ and $\varepsilon_y = 0.0625$. The left part of the figure lays out the discretized solution space, composed only of consistent (`white`) and inconsistent (`blue`) areas. The right part sets out the beginning of its hierarchical data structure.

Let us compute the consistency of the following nodes by using interval arithmetic:

- Let N_1 be the node defined by the pair of intervals $([0, 0.5], [0.5, 1])$. N_1 is `white` because it completely satisfies the constraint $\boldsymbol{C}(x,y)$: $d_n^y \ominus (d_n^x)^3 \geqslant \mathbf{0} \Leftrightarrow [0.475, 1] \geqslant [0, 0]$ (true for all the computed interval).
- Let N_2 be the node defined by the pair $([1, 2], [-1, 0])$. N_2 is `blue` because it does not satisfy the constraint $\boldsymbol{C}(x,y)$: $d_n^y \ominus (d_n^x)^3 \geqslant \mathbf{0} \Leftrightarrow [-9, -1] \geqslant [0, 0]$ (false for all the computed interval).
- Let N_3 be the node defined by the pair $([1, 2], [1, 2])$. N_3 is `grey` because it partially satisfies the constraint $\boldsymbol{C}(x,y)$: $d_n^y \ominus (d_n^x)^3 \geqslant \mathbf{0} \Leftrightarrow [-9, 1] \geqslant [0, 0]$ (true and false on the computed interval).

Algorithm 1, named Build Quad Tree, builds the Quad Tree of a continuous constraint $\boldsymbol{C}(x,y)$ from one of its nodes given as a parameter. In this algorithm, only the completely inconsistent regions have to be rejected, therefore the *unitary* `grey` nodes turn `white`.

4. Piecewise constraints and Quad Tree

The aim of this section is to extend the Quad Tree approach in order to be able to take into account piecewise constraints. A first sub-section defines what we mean by piecewise constraints and introduces what we call information grades. Following sub-sections present Quad Trees which deal with constraints assembling equalities, $f(x,y) = 0$, and inequalities, $f(x,y) >$, $\geqslant$, $\leqslant$ or < 0.

4.1. Piecewise constraints definition and example

Definition 3. A piecewise constraint $\boldsymbol{C}(x,y)$ is defined on a search area (Dx, Dy) and composed of a set of continuous constraints as defined by Definition 1, called pieces and noted $f_i(x,y)$. A piece $f_i(x,y)$ is defined on a particular domain $(D_x^{f_i}, D_y^{f_i})$ and is either an equality constraint $f_i(x,y) = 0$ or an inequality constraint $f_i(x,y) \odot 0$, with $\odot$ belonging to $\{<, \leqslant, >, \geqslant\}$. A piecewise constraint $\boldsymbol{C}(x,y)$ is either a set of equality constraints or a set of inequality constraints.

We notice that a region (d_X, d_Y) of the search area (Dx, Dy) can be covered by more than a single domain $(D_x^{f_i}, D_y^{f_i})$ or by none at all.

Some hypotheses on the general outline of piecewise constraints are necessary to guarantee the existence of a border between the consistent and inconsistent regions. These hypotheses are considered verified when the Quad Tree generation

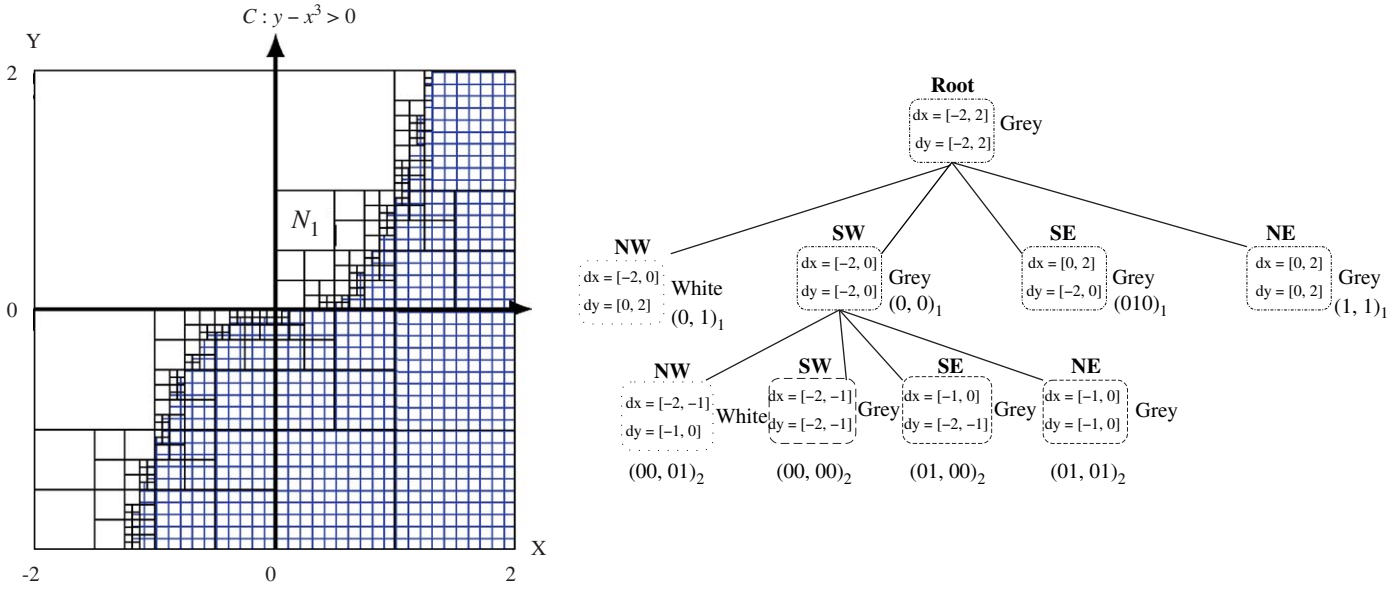

Fig. 4. Quad Tree of a continuous constraint.

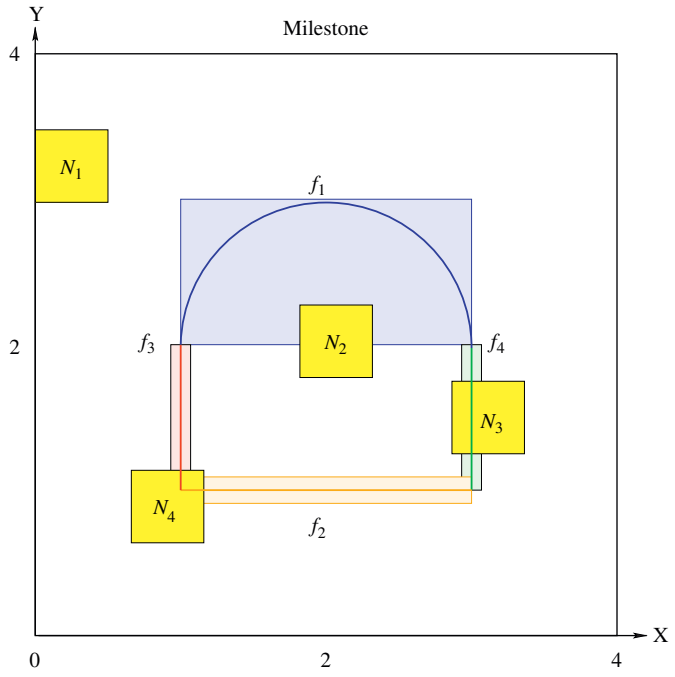

Fig. 5. Grades of information.

is launched:

- the general outline of the piecewise constraint $\boldsymbol{C}(x,y)$ must be closed in case of inequality constraints, and must be bounded in case of equality ones,
- in the case of inequality constraints, the pieces should not cross each other and all the pieces must be consistent with the others: the border between the consistent and inconsistent regions must be clearly defined.

Fig. 5 shows an example of piecewise constraints, named 'Milestone', composed of four pieces:

- $\boldsymbol{f}_1(x,y) : (x-2)^2 + (y-2)^2 \leqslant 1$, on the domain $D_x^{f_1} = [1,3], D_y^{f_1} = [2,3]$,
- $\boldsymbol{f}_2(x,y) : y \geqslant 1$, on the domain $D_x^{f_2} = [1,3], D_y^{f_2} = [0.9,1.1]$,
- $\boldsymbol{f}_3(x,y) : x \geqslant 1$, on the domain $D_x^{f_3} = [0.9,1.1], D_y^{f_3} = [1,2]$,
- $\boldsymbol{f}_4(x,y) : x \leqslant 3$, on the domain $D_x^{f_4} = [2.9,3.1], D_y^{f_4} = [1,2]$.

Let us look at the nodes N_1, N_2, N_3 and N_4:

- The node N_1 does not intersect any domain $(D_x^{f_i}, D_y^{f_i})$ of the pieces $f_i(x,y)$ and does not intersect any piece $f_i(x,y)$.
- The node N_2 intersects the domain $(D_x^{f_1}, D_y^{f_1})$ of $f_1(x,y)$, but it intersects none of the pieces $f_i(x,y)$.
- The node N_3 intersects the domain $(D_x^{f_4}, D_y^{f_4})$ of $f_4(x,y)$ and intersects this piece $f_4(x,y)$.
- The node N_4 intersects the domains $(D_x^{f_2}, D_y^{f_2})$ and $(D_x^{f_3}, D_y^{f_3})$, the first one corresponding to $f_2(x,y)$ and the second one to $f_3(x,y)$, and it intersects the two pieces $f_2(x,y)$ and $f_3(x,y)$.

4.2. Grades of information

As a node n of a Quad Tree can either intersect several domains $(D_x^{f_i}, D_y^{f_i})$ and/or several pieces $f_i(x,y)$, 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, firstly, to the intersection between a node n and the domain of a piece $(D_x^{f_i}, D_y^{f_i})$, and secondly, to the intersection between a node n and a piece itself $f_i(x,y)$. The previous intersections are computed by using interval analysis.

Four types of nodes are then characterized by the grade of information.

- Firstly, we have *empty* nodes which do not have any information to determine their consistency with the piecewise constraint $\boldsymbol{C}(x,y)$.

  Definition 4. A node n defined by (d_n^x, d_n^y) is an *empty* node if it does not intersect any domain $(D_x^{f_i}, D_y^{f_i})$ and any piece $f_i(x,y)$:

  $(d_n^x, d_n^y) \cap (D_x^{f_i}, D_y^{f_i}) = \emptyset$ for the domain of any piece
  $(d_n^x, d_n^y) \cap f_i(x,y) = \emptyset$ for any piece $f_i(x,y)$

- Secondly, we have *poorly informed* nodes which do not have enough information to determine their consistency with the piecewise constraint $\boldsymbol{C}(x,y)$.

Definition 5. A node n defined by (d_n^x, d_n^y) is a *poorly informed* node if, firstly, it intersects at least one domain $(D_x^{f_i}, D_y^{f_i})$ and secondly, does not intersect any piece $f_i(x,y)$:

$(d_n^x, d_n^y) \cap (D_x^{f_i}, D_y^{f_i}) \neq \emptyset$ at least for the domain of one piece
$(d_n^x, d_n^y) \cap f_i(x,y) = \emptyset$ for any piece $f_i(x,y)$

- Thirdly, we have *informed* nodes which have enough information to determine their consistency with the piecewise constraint $\boldsymbol{C}(x,y)$.

Definition 6. A node n is an *informed* node if it intersects one and only one piece $f_i(x,y)$, whatever the number of domains $(D_x^{f_i}, D_y^{f_i})$ it intersects:

$(d_n^x, d_n^y) \cap (D_x^{f_i}, D_y^{f_i}) \neq \emptyset$ at least for the domain of one piece
$(d_n^x, d_n^y) \cap f_i(x,y) \neq \emptyset$ for one, and only one piece $f_i(x,y)$

- And finally, we have *over-informed* nodes which are full of information to determine their consistency with the piecewise constraint $\boldsymbol{C}(x,y)$.

Definition 7. A node n is an *over-informed* node if it intersects more than one piece $f_i(x,y)$, whatever the number of domains $(D_x^{f_i}, D_y^{f_i})$ it intersects:

$(d_n^x, d_n^y) \cap (D_x^{f_i}, D_y^{f_i}) \neq \emptyset$ at least for the domain of one piece
$(d_n^x, d_n^y) \cap f_i(x,y) \neq \emptyset$ for more than one piece $f_i(x,y)$

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

If we consider the previous example, Fig. 5, we can characterize the type of the four nodes N_1, N_2, N_3 and N_4 by their grade of information as follows: the node N_1 an *empty* node, the N_2 is a *poorly informed* node, the node N_3 is an *informed* node, the N_4 is an *over-informed* node.

4.3. Generation of a Quad Tree of a piecewise constraint

In this subsection, the generation of Quad Trees of piecewise constraints is presented for constraints assembling inequalities and equalities.

4.3.1. Piecewise constraints assembling inequalities

The idea of the previous section, associating a grade of information to each node, is kept. But the generation of a Quad Tree relevant to a piecewise constraint assembling inequalities, is somewhat complex: *ignorant* nodes can belong either to consistent or inconsistent regions. Therefore, generation is achieved in two steps. Firstly, during recursive decomposition, we need to identify and mark the information grade of each node with a particular colour. Secondly, when decomposition stops, the consistency of each *ignorant* node is found, thanks to the propagation of the consistent and inconsistent regions.

The first step of Quad Tree generation is recursive, based on the grades of information and follows the principles below:

- each node n is defined by a pair of intervals (d_n^x, d_n^y),
- each node n has a specific code, corresponding to its coordinates,
- if a node n is an *empty* node, it is coloured `red`,
- if a node n is a *poorly informed* node, it is coloured `green`,
- if a node n is an *over-informed* one (more than one piece intersects the node), it is coloured:
  - `orange` if it is *unitary* (one of the decomposition steps, ε_x and ε_y, is reached),
  - `grey` otherwise and split into four children: NW, SW, SE and NE, each of whose grade of information has to be computed and marked,
- if a node n is an *informed* one (only one piece $f_i(x,y)$ intersects the node), it is coloured:
  - `yellow` if it is *unitary* (one of the decomposition steps, ε_x and ε_y, is reached),
  - `grey` otherwise and classical Quad Tree generation is launched from the isolated piece (cf. Section 3.1). A colour is given to the node:
    `white` if the node n is consistent with the isolated piece $f_i(x,y)$,
    `blue` if the node n is inconsistent with the isolated piece $f_i(x,y)$,
    `grey` if the node n is, at once, consistent and inconsistent with the isolated piece $f_i(x,y)$,
- two discretization steps, ε_x and ε_y, 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 Quad Tree can be coloured either `white` for consistent nodes, `blue` for inconsistent nodes, `yellow` for *unitary informed* nodes, `orange` for *unitary over-informed* nodes, `red` for *empty* nodes and `green` for *poorly informed* nodes.

In order to illustrate these different colours, let us consider the inside area and the border of the constraint named 'Milestone' of Fig. 5, as the consistent region. In this example, $\varepsilon_x = 0.25$ and $\varepsilon_y = 0.25$. When the first step is finished as shown in Fig. 6, the Quad Tree is multicoloured with `white`, `blue`, `red`, `green`, `yellow` and `orange`.

Let us look in detail at the decomposition of the node defined by the pair $(d_n^x = [1, 1.5], d_n^y = [1.5, 2])$ and encoded $(2,3)_3$. This node is an *over-informed* node intersecting f_1 and f_3: its colour is therefore `grey` and it must be split once more into four child nodes which are:

- a *unitary over-informed* node (NW), intersecting f_1 and f_3, for its first child, defined by the pair $(d_n^x = [1.25, 1.5], d_n^y = [1.75, 2])$ and encoded $(6,7)_4$: its colour is therefore `orange`,

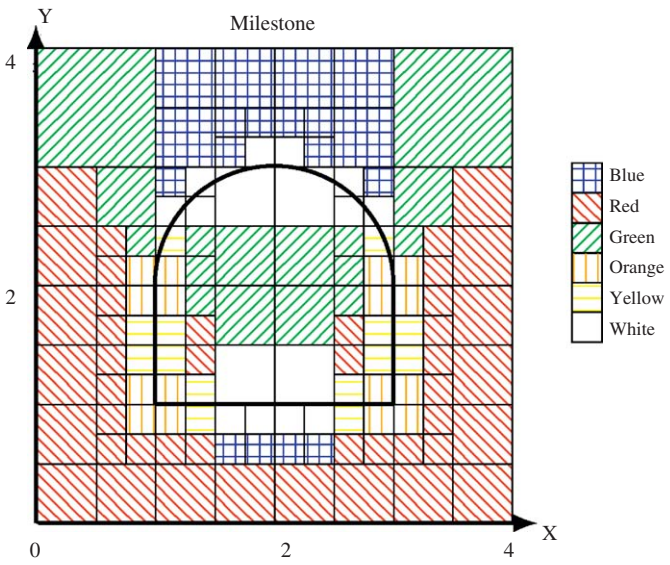

Fig. 6. Quad Tree at the end of the first step. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

- a *unitary informed* node (SW), intersecting only f_3, for its second child, defined by the pair ($d_n^x = [1.25, 1.5]$, $d_n^y = [1.5, 1.75]$) and encoded $(6,6)_4$: its colour is therefore `yellow`,
- a *unitary empty* node for its third child (*SE*), defined by the pair ($d_n^x = [1.5, 1.75]$, $d_n^y = [1.5, 1.75]$) and encoded $(7,6)_4$: its colour is therefore `red`,
- and a *unitary poorly informed* node (*NE*), intersecting the domain of f_1, for its fourth child, defined by the pair ($d_n^x = [1.5, 1.75]$, $d_n^y = [1.75, 2]$) and encoded $(7,7)_4$: its colour is therefore `green`.

The second step of Quad Tree generation consists in propagating the consistent and inconsistent regions from the nodes which know their consistency (the `yellow`, `blue` and `white` nodes), to those which are *ignorant* (the `red` and the `green` ones). In order to propagate the consistent and the inconsistent regions, the neighbours of the `yellow`, `blue` and `white` nodes must be identified. The selection of the relevant neighbours of a node n is made thanks to the 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. The detailed selection of the relevant neighbours of a node n can be found in Vareilles (2005).

The propagation of consistent and inconsistent regions is done in four steps. In the first place, `yellow` nodes indicate to their `red` and `green` neighbours on which side of the border they belong, by using interval arithmetic applied to the isolated piece their are intersecting. If they belong to the consistent side, `red` and `green` neighbours turn `white` or if not, `blue`. Let us notate Border the algorithm which colours the `red` and `green` nodes in `white` or `blue` depending on the side of the border they belong to. Fig. 7 illustrates this first step on the piecewise constraint named 'Milestone'.

For instance, let us look at the `yellow` node, defined by the pair ($d_n^x = [1, 1.25], d_n^y = [2.25, 2.5]$) and encoded $(4,9)_4$. We can see that this node has told its `green` neighbours on its left that it belongs to the inconsistent side: therefore, the `green` neighbour on the left turns `blue`, and it has told its `green` neighbour on its right, that it belongs to the consistent side: therefore, the `green` neighbour on the right turns `white`.

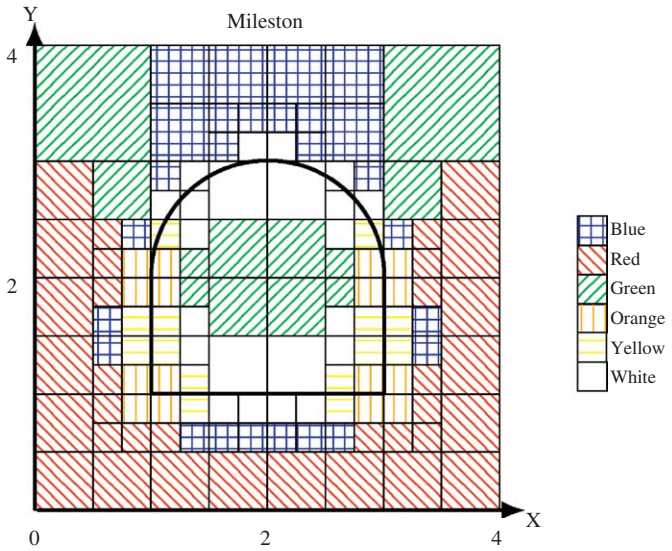

Fig. 7. Propagation from the `yellow` nodes to their `red` and `green` neighbours. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

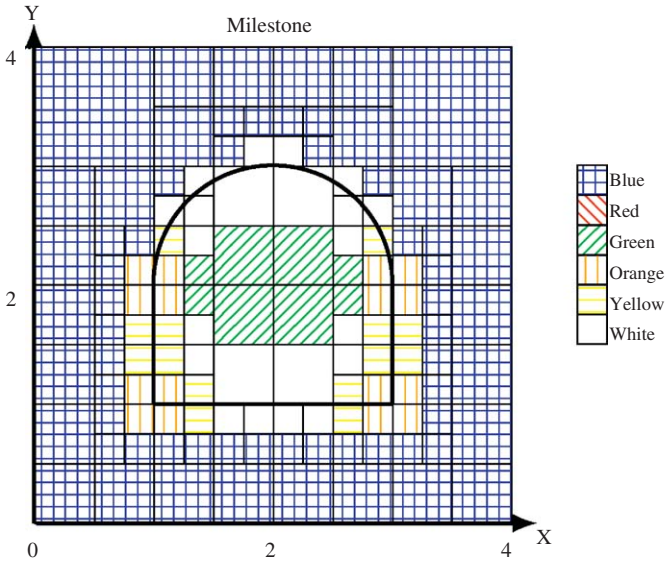

Fig. 8. Propagation from the `blue` nodes to their `red` and `green` neighbours. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In the second place, `blue` nodes tell their *ignorant* neighbours (the `red` and `green` ones) that they belong to the inconsistent region, therefore they turn `blue`. Let us notate Turn blue the algorithm which propagates the inconsistent region to the *ignorant* nodes. Fig. 8 shows the propagation of the inconsistent region on the constraint 'Milestone'.

In the third place, `white` nodes tell their *ignorant* neighbours that they belong to the consistent region, therefore they become `white`. Let us notate Turn white the algorithm which propagates the consistent region on the *ignorant* nodes. Fig. 9 shows the propagation of the consistent region on the piecewise constraint 'Milestone'.

Finally, `yellow` and `orange` nodes turn either `white` if we choose to keep the border within the consistent region or `blue` if only the regions which are completely legal with the piecewise constraint need to be kept. Let us notate Outline the algorithm which colours the `yellow` and `orange` nodes in `white` or `blue` depending on the case we are studying. Fig. 10 illustrates the coloration of `yellow` and `orange` nodes in `white` on the piecewise constraint 'Milestone'.

The generation of the Quad Tree relevant to a piecewise constraint assembling inequalities needs different algorithms:

- Build inequality Quad Tree, presented by the Algorithm 2, which divides the sub-space of a node n into four child nodes, whose grade of information has to be found,
- Build Quad Tree, presented by the Algorithm 1, which builds the Quad Tree of a continuous constraint passed as parameter,
- Border which colours the `red` and `green` nodes in `white` or `blue` depending on the side of the border they belong to,
- Turn blue which propagates the inconsistent regions on `red` and `green` nodes,
- Turn white which propagates the inconsistent regions on `red` and `green` nodes,
- Outline which colours the `yellow` and `orange` nodes in white.

The detailed algorithms Border, Turn blue, Turn white and Outline can be found in Vareilles (2005).

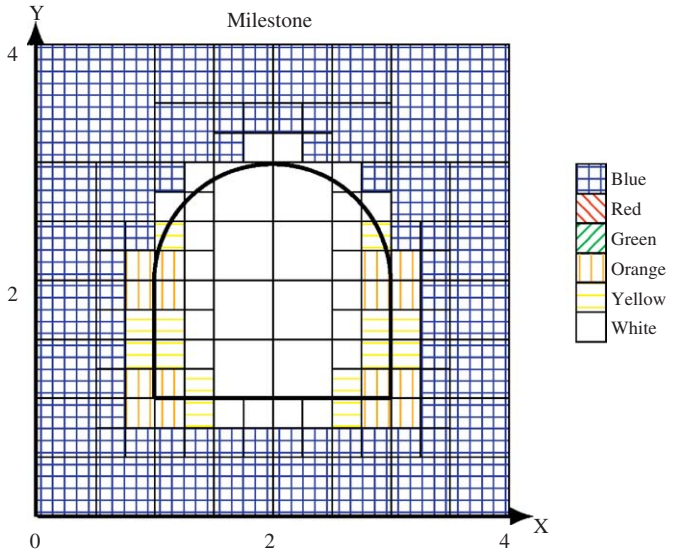

Fig. 9. Propagation from the `white` nodes to their `red` and `green` neighbours. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

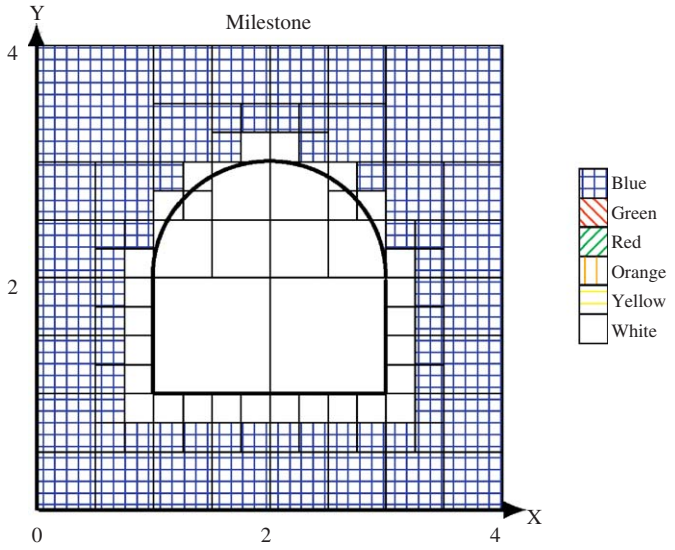

Fig. 10. Colouration of the `yellow` and `orange` nodes in `white`. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4.3.2. Piecewise constraints assembling equalities

The generation of Quad Trees of piecewise constraints assembling equalities is easier, because the consistent region is defined by the path of the piecewise constraint itself. Therefore, the generation is achieved in only one step: during recursive decomposition, we colour directly *ignorant* nodes in `blue`, because they always belong to the inconsistent regions, *informed* nodes are coloured `grey` and classical Quad Tree generation is launched from the isolated piece and *over-informed* nodes are coloured `grey` and split into four children. Only *informed* and *over-informed unitary* nodes are coloured `white`.

5. Conclusion

The aim of this paper has been to propose an approach that permits us to handle binary piecewise constraints in *CSPs*. Quad Trees, proposed by Sam (1995), allow single continuous constraint to be taken into account thanks to a recursive decomposition of the search area into nodes, whose consistency with the constraint is computed and marked with particular colours.

We have extended this method to piecewise constraints, with respect to some insubstantial assumptions on their general outline. Our method follows the idea of classical Quad Trees (recursive decomposition) but the generation of Quad Trees of piecewise constraints is somewhat more complex. We have to identify the grade of information of each node. Indeed, some regions can be covered by several domains of the pieces and/or by several pieces, or by none at all. These kinds of areas cannot alone determine their consistency with the piecewise constraint. Four grades of information have been identified to characterize the different types of nodes. In the case of piecewise constraints assembling inequalities, 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 filtering methods and the mechanism of 'fusion' proposed by Sam (1995), to compute the intersection of constraints represented by Quad Trees, can be applied to Quad Trees associated to piecewise constraints, without any particular problems. Indeed, these mechanisms work on the `white` and `blue` nodes of the resulting Quad Trees. 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.

Algorithm 1. Build Quad Tree(Node: n)

— ⋱ — ***This algorithm builds the Quad Tree*** $T_{(x,y)}$ ***of the constraint*** $\boldsymbol{C}(x,y)$ ***from one of its nodes passed as parameter***.
If (None of the decomposition steps is reached) **Then**
  The node n turns `grey` and is split into four child nodes
  For each of the children s of n **Do**
    Encoding of the node s
    If (s is completely inconsistent with $\boldsymbol{C}(x,y)$) **Then**
      The node s turns `blue`
    Else
      If (s is completely consistent with $\boldsymbol{C}(x,y)$) **Then**
        The node s turns `white`
      Else
        — ⋱ — *The node s is partially consistent with* $\boldsymbol{C}(x,y)$*: it must decomposed once more*
        $s \leftarrow$ Build Quad Tree(s) (Algorithm 1)
      End If
    End If
  End For
Else
  — ⋱ — *The node n is* ***unitary***
  If (n is partially consistent with $\boldsymbol{C}(x,y)$) **Then**
    The node n turns `white`
  Else
    The node n is completely inconsistent and turns `blue`
  End If
End If
Return the current node

Algorithm 2. Build inequality Quad Tree(Node : n)

— ⋱ — ***This algorithm builds the Quad Tree*** $T_{(x,y)}$ ***of the constraint*** $\boldsymbol{C}(x,y)$ ***from one of its nodes passed as parameter***.
While (None of the decomposition steps is reached) **Do**
  Creation of the four child nodes of n
  For each of the children s of the current node n **Do**
    Pieces_Dom list of all the pieces $\boldsymbol{c}_i$ whose domain intersects the node f : $d_f^x \cap D_x^{c_i} \neq \emptyset \wedge d_f^y \cap D_y^{c_i} \neq \emptyset$
    If (*Pieces_Dom* $= 0$) **Then**
      — ⋱ — *The node s is an empty node*
      The node s turns `red`

Else
— ⋱ — *At least one domain of a piece $f_i(x, y)$ intersects the node s.*
If (the node *s* is an *informed* node) **Then**
— ⋱ — *We have to generate the Quad Tree associated to isolated piece f_i on the sub-space of the node (d_s^x, d_s^y)*
$s \leftarrow$ BUILD QUAD TREE(s) (Algorithm 1 on the proceeding page)
Else
If (The node *s* is an *over-informed* nodes) **Then**
— ⋱ — *The node s must be decomposed once more*
$s \leftarrow$ BUILD INEQUALITY QUAD TREE(s) (Algorithm 2)
Else
— ⋱ — *The node s is a* poorly-informed *node*
The node *s* turns `green`
End If
End If
End If
End For
End While
If (The node *n* is a *unitary informed* node (only one piece $f_i(x, y)$ intersects it))
Then
The node turns `yellow`
Else
If (The node *n* is a *unitary over-informed* node (several pieces $f_i(x, y)$ intersect it))
Then
The node turns `orange`
Else
If (The node *n* is a *unitary poorly informed* node) **Then**
The node turns `green`
Else
The node is a *unitary empty* node: it turns `red`
End If
End If
End If
Return *n*

References

Aldanondo, M., Vareilles, E., Lamesle, P., Hadj-Hamou, K., Gaborit, P., 2005. Interactive configuration and evaluation of a heat treatment operation. In: Workshop on Configuration. International Joint Conference on Artificial Intelligence.

Benhamou, F., McAllester, D., van Hentenryck, 1994. CLP(intervals) revisited. In: Proceedings of the International Logic Programming Symposium.

Briggs, J., Peat, D., 1991. Un miroir turbulent—Guide illustré de la théorie du chaos. Harper and Row publishers (translated by D. Stoquart).

Chenouard, R., Sebastian, P., Granvilliers, L., 2007. Solving an Air Conditioning System Problem in an Embodiment Design Context Using Constraint Satisfaction Techniques, CP, pp. 18–32.

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

Lamesle, P., Vareilles, E., Aldanondo, P., 2005. Towards a KBS for a qualitative distortion's prediction for heat treatments. In: International Conference on Distortion Engineering.

Lhomme, O., 1993. Consistency technique for numeric CSP. In: International Joint Conference on Artificial Intelligence, pp. 283–311.

Lottaz, C., 2000. Collaborative design using solution spaces. Ph.D. Thesis, École Polytechnique Fédérale de Lausanne.

Moore, R.E., 1966. Interval Analysis. Prentice-Hall, Englewood Cliffs, NJ.

Sam, D., 1995. Constraint consistency techniques for continuous domains. Ph.D. Thesis, École Polytechnique Fédérale de Lausanne.

Samet, H., 1984. The quadtree and related hierarchical structures. Computing Surveys, 187–260.

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

Vareilles, E., Aldanondo, M., Gaborit, P., 2007. Evaluation and design: a knowledge based approach. International Journal of Intelligent Manufacturing 20 (7), 639–653.