The Octagon Abstract Domain for Continuous Constraints

Marie Pelleau, Charlotte Truchet, Frédéric Benhamou

LINA, UMR CNRS 6241
Université de Nantes, France
2 rue de la Houssinière, Nantes, France
firstName.lastName@univ-nantes.fr

Abstract. Domains in Continuous Constraint Programming are generally represented with intervals whose n-ary Cartesian product (box) approximates the solution space. This paper defines a generic solver and proposes a new representation for continuous variable domains based on octagons. We generalize local consistency and split to this octagon representation. Preliminary experimental results show promising performance improvements on several classical benchmarks.

1 Introduction

Continuous Constraint Programming relies on interval representation of the variables domains. Filtering and solution set approximations are based on Cartesian products of intervals, called boxes. In this paper, we propose to improve the Cartesian representation precision by introducing an $n-$ary octagonal representation of domains in order to improve filtering accuracy.

By introducing non-Cartesian representations for domains, we do not modify the basic principles of constraint solving. The main idea remains to reduce domains by applying constraint propagators that locally approximate constraint and domains intersections (filtering), by computing fixpoints of these operators (propagation) and by splitting the domains to search the solution space. Nevertheless, each of these steps has to be redesigned in depth to take the new domains into account, since we lose the convenient correspondence between approximate intersections and domain projections. In this article, we propose generic definitions to entirely abstract the domains representation from the Constraint Programming solving process. All steps are redesigned so that any representation can be used, under some conditions, in the same way Abstract Interpretation in Semantics uses abstract domains.

Abstract Interpretation is a domain of Semantics dealing with program proofs [1]. Program properties on the values of the variables are proven by abstracting the sets of these values, and manipulating the abstractions, instead of the variables. The abstractions can be of different shapes (boxes, polyhedrons, ellipsoids...) and are called abstract domains. An important point is that the shape of the domains is a parameter of the program prover, in other words, a program

can be proved using any abstract domain without changing the program prover. Depending on the program properties to analyze, abstract domains can be more or less efficient. In particular, relational abstract domains (where relations between variables are taken into account) have been proven to be very powerfull for analyzing continuous variables [2]. We use the same idea in Constraint Programming.

While shifting from a Cartesian to a generic approach, the resolution process is very similar. In the interval case, one starts with the Cartesian product of the initial domains and propagators reduce this global box until reaching a fixpoint. In the generic case, the Cartesian product of the initial domains is itself an abstract domain and each constraint propagator computes in turn the smallest abstract domain containing the intersection of the global abstract domain and the constraint itself, until reaching an abstract domain fixpoint. In both cases, splitting operators drive the search space exploration, alternating with global domain reduction.

An example of this generic solver is given with the octagons as the domain representation. The octagon are chosen for different reasons: they represent a reasonable tradeoff between boxes and more complex approximation shapes (*e.g.,* polyhedron, ellipsoids) and they have been studied in another context to approximate numerical computations in static analysis of programs. More importantly, we show that octagons allows us to translate the corresponding constraint systems in order to incorporate classical continuous constraint tools in the resolution.

The contributions of this paper concern the different aspects of abstract domain solving. First, we show how to define abstract domains in Constraint Programming. This is done by defining generically different operators such as the consistency and the splitting operator. Second, we give an example of our generic solver: the octagonal solver. We thus propose a split algorithm adapted to the octagonal case, define a specific local consistency and propose an appropriate algorithm, built on top of any continuous filtering method. Finally, we present another domain representation based on the octagons: the partial octagon abstract domain. The main idea is to offer a new tradeoff between octagons and intervals. Different heuristics to create partial octagons are presented.

2 Preliminaries

This section presents some of the notions in Abstract Interpretation upon which the Abstract Domains are based. The abstract domains live in lattices. We recall here some definitions related to this notion and illustrate them with examples from Continuous Constraint Programming. We recall some necessary background of Continuous Constraint Programming such as consistency. A full presentation of Abstract Interpretation is presented in [1] and of Constraint Programming in [3].

2.1 Lattices

Lattice is a well known notion in computer science. Here they are used to express some operations on abstract domains, and required to have the appropriate properties: closure, completeness.

Definition 1 (Poset). *A relation $\sqsubseteq$ on a non-empty set E is a* partial order *(po) iff it is reflexive, antisymmetric and transitive. A set with a partial order is called a* partially ordered set *(poset).*

Example 1. Consider $\mathbb{F}$ the set of floating point numbers according to the IEEE norm [4]. For $a, b \in \mathbb{F}$, let $[a, b] = \{x \in \mathbb{R}, a \leq x \leq b\}$ the real interval delimited by a and b, and $\mathbb{I} = \{[a, b], a, b \in \mathbb{F}\}$ the set of all such intervals. For any interval I, we write $\underline{I}$ (resp. $\overline{I}$) its lower (resp. upper) bound. Similarly, for any real point x, $\underline{x}$ is the floating-point lower approximation of x (resp. $\overline{x}$, upper).

Let $\mathbb{I}^n$ the set of all Cartesian products of n intervals. The set $\mathbb{I}^n$ with the relation $\subset$ is a poset.

Remark 1. Note that the relation $\subset$ is a partial order. Thus any non-empty set with this relation forms a poset.

Definition 2 (Lattice). *A partially ordered set $(E, \sqsubseteq, \sqcup, \sqcap)$ is a* lattice *iff for $a, b \in E$, the pair $\{a, b\}$ has both a least upper bound (lub), denoted by $a \sqcup b$, and a greatest lower bound (glb), denoted by $a \sqcap b$. It is* complete *iff any set has both a lub and a glb.*

A complete lattice has a least element and a greatest element.

Example 2. It is easily checked that $(\mathbb{I}^n, \subset)$ with the least element $\bot = \emptyset$ and the greatest element $\top = \mathbb{R}^n$ is a complete lattice. Let $D = D_1 \times \cdots \times D_n, D' = D'_1 \times \cdots \times D'_n$ be any two elements in $\mathbb{I}^n$. Then the pair $\{D, D'\}$ has both a *lub* $D \cap D' = [\max(\underline{D_1}, \underline{D'_1}), \min(\overline{D_1}, \overline{D'_1})] \times \cdots \times [\max(\underline{D_n}, \underline{D'_n}), \min(\overline{D_n}, \overline{D'_n})]$ and a *glb* $D \cup D' = [\min(\underline{D_1}, \underline{D'_1}), \max(\overline{D_1}, \overline{D'_1})] \times \cdots \times [\min(\underline{D_n}, \underline{D'_n}), \max(\overline{D_n}, \overline{D'_n})]$. Any set has thus a *lub* and a *glb* , therefore $(\mathbb{I}^n, \subset)$ is a complete lattice.

Lattices are the base set for the Abstract Domains in Abstract Interpretation. An important feature of Abstract Domains is that they can be linked by Galois connections:

Definition 3 (Galois Connection). *Given two posets E_1 and E_2, a Galois connection is defined by two morphisms $\alpha : E_1 \to E_2$ and $\gamma : E_2 \to E_1$ such that*

- *α and γ are monotonic,*
- *$\forall X_1 \in E_1, X_2 \in E_2, \alpha(X_1) \sqsubseteq X_2 \iff X_1 \sqsubseteq \gamma(X_2)$.*

Notice that $(\alpha \circ \gamma)(X_2) \sqsubseteq X_2$ and $X_1 \sqsubseteq (\gamma \circ \alpha)(X_1)$: Galois connections do not lose elements.

A Galois connection links two posets such that each element of the first poset corresponds to an element in the second, and reciprocally. Note that the

element in the second poset is an overapproximation of the element in the first poset. It is often used in continuous constraints solving without being named. Indeed, as the intervals with reals bounds are not computer representable, they are approximated by floating-point intervals. The transformation from one to the other is a Galois connection as shown in the example below.

Example 3. Let $\mathbb{J}$ the set off all intervals with real bounds. Given the two posets $(\mathbb{I}^n, \subset)$ and $(\mathbb{J}^n, \subset)$, there exists a Galois connection such that

- $\alpha : \mathbb{J}^n \to \mathbb{I}^n, I \mapsto [\underline{a_1}, \overline{b_1}] \times \cdots \times [\underline{a_n}, \overline{b_n}]$, with $I = I_1 \times \cdots \times I_n$ and $I_i = [a_i, b_i], i \in \{1 \dots n\}$
- $\gamma : \mathbb{I}^n \to \mathbb{J}^n, I \mapsto I$ this is straightforward as a floating-point number is a real.

This subsection has set some notions needed for the abstract domain. We now recall some necessary background on Continuous Constraint Programming.

2.2 Constraint Programming

We consider a Constraint Satisfaction Problem (CSP) on variables $\mathcal{V} = (v_1 \dots v_n)$, taking their values in domains $\mathcal{D} = (D_1 \dots D_n)$, with constraints $(C_1 \dots C_p)$. The set of tuples representing the possible assignments for the variables is $D = D_1 \times \cdots \times D_n$. The solutions of the CSP are the elements of D satisfying the constraints. We denote by $\mathcal{S}$ the set of solutions, $\mathcal{S} = \{(s_1 \dots s_n) \in \mathcal{D}, \forall i \in 1..p, C_i(s_1 \dots s_n)\}$.

Definition 4 (Approximation). *A complete approximation (resp. sound approximation), of the solution set is a sequence $D'_1 \dots D'_n$ such that $D'_i \subset D_i$, and $\mathcal{S} \subset D'_1 \times \cdots \times D'_n$ (resp. $D'_1 \times \cdots \times D'_n \subset \mathcal{S}$).*

Soundness guarantees that the elements are solutions, while completeness guarantees that no solution is lost. In practice, a constraint solver on finite domains is required to be sound and complete, which is equivalent to compute the solutions. On continuous domains, the fact that the reals are not computer representable make things more complicated, and one usually has to withdraw either soundness (most of the time) or completeness.

A very important notion in Constraint Programming is that of consistency. Several variants of consistency have been proposed, depending on the type of the domains and other practical considerations. The most usual for continuous constraints is the following.

Definition 5 (Box). *A cartesian product of intervals is called a box. The set of all the boxes is denoted by $\mathbb{B}$.*

Definition 6 (Hull-Consistency). *Let $v_1 \dots v_n$ be variables over continuous domains represented by intervals $D_1 \dots D_n \in \mathbb{I}$ and C a constraint. The domains $D_1 \dots D_n$ are said Hull-consistent for C iff $D_1 \times \cdots \times D_n$ is the smallest floating-point box containing the solutions for C.*

This definition is for one constraint, for a system of constraints it is sufficient to apply the consistency for each constraint recursively until a fixpoint is reached. This is call the propagation. It has been shown, in [5] by Apt or in [6], that the order in which the consistencies were applied does not matter. This comes from the fact that the propagation scheme lives in a complete finite lattice and thus any set has a least element: the consistent fixpoint.

Even if Constraint Programming and Abstract Interpretation have different goals they both rely on the same theoretical background: lattices and fixpoints. While in continuous Constraint Programming we use Cartesian product of interval to represent the domains, in Abstract Interpretation they have defined several domain representation called Abstract Domains such as zones, octagons, ellipsoids to name a few. See [7] for a panel of different existing abstract domains.

The next section presents how to benefit from the work done in Abstract Interpretation on abstract domains.

3 Abstract Domains for Constraints

In this section, we define the abstract domains in Constraint Programming. This will allow us to use them as domain representation during the solving process.

3.1 Consistency as lattice fixpoints

Let E be a subset of $\mathcal{P}(\mathcal{D})$, with inclusion as a partial order. We will write E^f for $E \backslash \emptyset$. In the following, we will restrain the definitions to the case when E is closed by intersection, as this is sufficient for all the classical cases. If E is closed by intersection then it has a *glb* and is a directed lattice.

Definition 7 (E-Consistency). *Consider a constraint C. An element e is E-consistent for C iff it is a least element for $\mathcal{C}_C^E = \{e' \in E, \mathcal{S}_C \subset e'\}$. If E is closed under intersection, $\mathcal{C}_C^E$ is a complete lattice and there exists a unique E-consistent element for C in E. In that case, this element is written $\mathbf{C}_C^E$.*

Proof. Let $\mathbf{C}_C^E = \bigcap_{e \in E, \mathcal{S}_C \subset e} e$. We first prove that $\mathbf{C}_C^E$ is the least element for $\mathcal{C}_C^E$. Let $A \in \mathcal{C}_C^E$ strictly smaller than $\mathbf{C}_C^E$, thus $\mathcal{S}_C \subset A$. As $\mathbf{C}_C^E$ is the intersection of all such elements, A can not be strictly smaller than $\mathbf{C}_C^E$. Hence, $\mathbf{C}_C^E$ is the least element of $\mathcal{C}_C^E$.

We now prove that $\mathbf{C}_C^E$ is unique. Assume that there exists and element $B \in \mathcal{C}_C^E$ such that B is also a least element for $\mathcal{C}_C^E$. If such an element exists, then it contains all the solutions, $\mathcal{S}_C \subset B$. As $\mathbf{C}_C^E$ is the intersection of all the elements containing $\mathcal{S}_C$, then $\mathbf{C}_C^E \subset B$. Hence, $\mathbf{C}_C^E$ is the unique least element of $\mathcal{C}_C^E$. □

Definition 8. *Let $e \in E$. It is E-consistent for $C_1 \dots C_p$ iff it is a least element for $\mathcal{C}_{C_1 \wedge \dots \wedge C_p}^E = \{e' \in E, \mathcal{S}_{C_1 \wedge \dots \wedge C_p} \subset e'\}$. If such an element exists, it is written $\mathbf{C}_{C_1 \wedge \dots \wedge C_p}^E$. If there is no ambiguity $\mathcal{C}_{C_1 \wedge \dots \wedge C_p}^E$ is written $\mathcal{C}^E$ and its least element $\mathbf{C}^E$.*

Proposition 1. *If E is closed by intersection then $\mathbf{C}^E$ exists and is unique. In addition, the set of all $\mathbf{C}^E_{C_{i_1}\wedge\cdots\wedge C_{i_k}}$ for $i_1 \dots i_k \in \{1 \dots p\}$ forms a lattice for inclusion and $\mathbf{C}^E_{C_1\wedge\cdots\wedge C_p}$ is its least element.*

Proof. The existence and the unicity of $\mathbf{C}^E$ comes from Definition 7.

We first prove that the set of all $\mathbf{C}^E_{C_{i_1}\wedge\cdots\wedge C_{i_k}}$ for $i_1 \dots i_k \in \{1 \dots p\}$ forms a lattice for inclusion. Let $\mathbf{C}^E_{C_i}$ and $\mathbf{C}^E_{C_j}$ for $i, j \in \{1 \dots p\}$ be any two elements of the set. Then the pair $\{\mathbf{C}^E_{C_i}, \mathbf{C}^E_{C_j}\}$ has both a *lub* $\mathbf{C}^E_{C_i} \cap \mathbf{C}^E_{C_j} = \bigcap_{e\in E, \mathcal{S}_{C_i}\subset e \vee \mathcal{S}_{C_j}\subset e} e$ and a *glb* $\mathbf{C}^E_{C_i} \cup \mathbf{C}^E_{C_j} = \bigcap_{e\in E, \mathcal{S}_{C_i}\subset e, \mathcal{S}_{C_j}\subset e} e$. Hence, the set of all $\mathbf{C}^E_{C_{i_1}\wedge\cdots\wedge C_{i_k}}$ for $i_1 \dots i_k \in \{1 \dots p\}$ is a lattice.

We now prove that $\mathbf{C}^E_{C_i\wedge C_j}$ is the least element for $\mathbf{C}^E_{C_i}, \mathbf{C}^E_{C_j}$, that is $\mathbf{C}^E_{C_i\wedge C_j} \subset \mathbf{C}^E_{C_i} \cap \mathbf{C}^E_{C_j}$. Let us first prove that $\mathbf{C}^E_{C_i\wedge C_j} \subset \mathbf{C}^E_{C_i}$. As $\mathcal{S}_{C_i\wedge C_j} = \{(s_1 \dots s_n) \in \mathcal{D}, C_i(s_1 \dots s_n) \wedge C_j(s_1 \dots s_n)\}$ and $\mathcal{S}_{C_i} = \{(s_1 \dots s_n) \in \mathcal{D},\ C_i(s_1 \dots s_n)\}$, then $\mathcal{S}_{C_i\wedge C_j} \subset \mathcal{S}_{C_i}$. Thus $\bigcap_{e\subset E, \mathcal{S}_{C_i\wedge C_j}\subset e} e \subset \bigcap_{e\subset E, \mathcal{S}_{C_i}\subset e} e$. Hence $\mathbf{C}^E_{C_i\wedge C_j} \subset \mathbf{C}^E_{C_i}$. Similarly, we prove that $\mathbf{C}^E_{C_i\wedge C_j} \subset \mathbf{C}^E_{C_j}$. As $\mathbf{C}^E_{C_i\wedge C_j} \subset \mathbf{C}^E_{C_i}$ and $\mathbf{C}^E_{C_i\wedge C_j} \subset \mathbf{C}^E_{C_j}$, then $\mathbf{C}^E_{C_i\wedge C_j} \subset \mathbf{C}^E_{C_i} \cap \mathbf{C}^E_{C_j}$. Therefore $\mathbf{C}^E_{C_i\wedge C_j}$ is the least element for $\mathbf{C}^E_{C_i}, \mathbf{C}^E_{C_j}$. □

This proposition is rather formal but still very useful: provided that each constraint of the CSP comes with a propagator, it is sufficient to apply these propagators recursively, no matter the order, until the fixpoint is reached, to achieve consistency for the CSP. On existing consistencies, a similar idea has been proposed in [5] by Apt or in [6].

We show below that the main existing continuous consistency is included by the definition.

Proposition 2. *The $\mathbb{I}^n$-consistency is Hull-consistency.*

Proof. We first prove that $\mathbb{I}^n$-consistent $\Longrightarrow$ Hull-consistent. Let $I \in \mathbb{I}^n$ be $\mathbb{I}^n$-consistent for a constraint C. Then I is the least element of $\mathcal{C}^{\mathbb{I}^n}_C$ which corresponds to the smallest Cartesian product of intervals with floating-point bounds containing all the solutions for C. Hence, by Definition 6, it is Hull-consistent.

We now prove that Hull-consistent $\Longrightarrow$ $\mathbb{I}^n$-consistent. Let $D = D_1 \times \cdots \times D_n$ Hull-consistent for a constraint C. It obviously contains all the solutions, we need to prove that it is the smallest such element of $\mathbb{I}^n$. Let $D' \in \mathbb{I}^n$, strictly smaller than D, then there is a i such that $D'_i \subsetneq D_i$. Let $I = D_i \backslash D'_i$, since $I \in D_i$ and D is Hull-consistent there also exist $I_j \in D_j$ for $j \neq i$ such that $C(I_1, \dots, I, \dots, I_n)$. This solution is not in D' and thus all $D' \subsetneq D$ lose solutions.

The $\mathbb{I}^n$-consistency is thus the Hull-consistency. □

The definition of consistency thus generalizes the existing consistency. For any $E \subset \mathcal{P}(D)$ closed by intersection (Octagons for instance), consistency is well-defined.

3.2 Split

Once the consistent element computed, we still need to explore it in order to find the solutions. This is done by recursively cutting the remaining search space into smaller elements, on which the consistency is then applied. An element is cut using a splitting operator as defined below.

Definition 9 (Splitting Operator). *Let $(E, \subset)$ be a poset. A splitting operator is an operator $\oplus : E \to \mathcal{P}(E)$ such that $\forall e \in E$,*

- $|\oplus(e)|$ *is finite,* $\oplus(e) = \{e_1 \dots e_k\}$,
- $\cup_{1 \leq j \leq k} e_j = e$,
- $\forall j \in \{1 \dots k\}$, $e \neq \emptyset \Longrightarrow e_j \neq \emptyset$,
- $e_j = e \Longrightarrow e$ *is a least element of* E^f.

The first condition is needed for the search process to terminate (finite width of the search tree). The second condition enforces that splitting does not loose solutions. We could have asked $\oplus(e)$ to be a partition of e, but this would forbid interesting possibilities and it is not mandatory to prove completeness of the solving process. The third condition is merely technical, to keep the fact that empty domains characterize inconsistency. The fourth condition means that the splitting operator actually splits: it is forbidden to keep the same domains.

Remark 2. It is important to notice that the definition implies: if e is not a least element of E^f, $e_k \subsetneq e$.

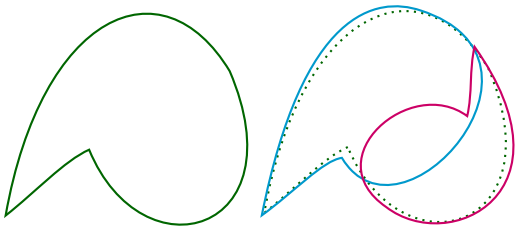
We show below that the continuous split is included in the definition.

Example 4. The continuous split on one variable interval is a splitting operator on $\mathcal{P}(\mathbb{I}) : \oplus_{\mathbb{I}}(I) = \{I^{\vdash}, I^{\dashv}\}$, where $I^{\vdash}$ and $I^{\dashv}$ are non empty subintervals of I with whatever way of managing the rounding errors on floats, provided that $I^{\vdash} \cup I^{\dashv} = I$. For the traditional splitting process for continuous CSP, given $I = [a, b]$, one usually return $I^{\vdash} = [a, \overline{\frac{a+b}{2}}]$ and $I^{\dashv} = [\underline{\frac{a+b}{2}}, b]$.

Example 5. The usual continuous split on a Catersian product of intervals is a splitting operator on $\mathcal{P}(\mathbb{I}^n)$. Let $D \in \mathbb{I}^n, D = D_1 \times \dots \times D_n$, the splitting operator first chooses a variable interval $D_i, i \in \{1 \dots n\}$. Then splits this interval using $\oplus_{\mathbb{I}}$. Which gives us: $\oplus_{\mathbb{I}^n}(D) = \{D_1 \times \dots \times D_i^{\vdash} \times \dots \times D_n, D_1 \times \dots \times D_i^{\dashv} \times \dots \times D_n\}$, where $\{D_i^{\vdash}, D_i^{\dashv}\} = \oplus_{\mathbb{I}}(D_i)$. For the traditional splitting process in continuous Constraint Programming, one usually splits the domain that realizes the maximum of $\max(\overline{D_i} - \underline{D_i})$.

In the following we write, for any E, $\oplus_{E,i}$ the splitting operator applied to the i-th variable.

With these generic definitions of the consistency and the splitting operator, we can define an abstract domain in Constraint Programming.



(a) a Shadok

(b) an example of the split on the Shadok abstract domain

Fig. 1. One can define a Shadok abstract domain.

3.3 Abstract Domain

Our goal is to define a generic solver with any domain representation. Here we define the abstract domain for Constraint Programming such that they have the requirements to be part of a solving process.

Definition 10 (Abstract Domain for Constraints). *An abstract domain is given by:*

- *a complete lattice E,*
- *a Galois connection between the variables domains and E,*
- *a normal form computer representable,*
- *a sequence of splitting operators on E,*
- *a strictly decreasing function $\tau : E \to \mathbb{R}$. s.t. $\tau(e) = 0 \Leftrightarrow e = \emptyset$*

The Galois connection links the abstract domains to the computer representation of the CSP domains. The normal form ensures that the abstract domain has a computational representation. The function τ represents some measure on the abstract domains. It expresses the precision of the abstract domain. It will be used to write the termination condition of the solving process.

Abstract domains can be defined independently from the domains of a particular CSP. They are intended to represent the shape of the domains representation. Of course, they can be cartesian, but this is no longer mandatory. Note that we do not formalize the propagators, because they depend both on the constraints and on the shape of the abstract domain. They have to be *ad hoc*.

One can, for instance, define a Shadok[1] abstract domain, given that they find a way to compute it. Figure 1 shows an example of a Shadok and a possible splitting operator.

With this definition of the abstract domains, we can define a solver based on abstract domains.

[1] http://www.lesshadoks.com/ or http://en.wikipedia.org/wiki/Les_Shadoks

```
int j ← 0 /*j indicates the depth in the search tree*/
int op[] /*at a depth j, stores the index of the chosen splitting operator, initialized uniformly at 0*/
int width[] /*at a depth j, stores the width of the tree already explored, initialized uniformly at 0*/
abstract domain e ∈ E /*any data structure to store the abstract domains*/
abstract domain sol[] /*stores the solutions*/
int nbSol ← 0 /*number of solutions*/

e = α(D) /*initialization to the abstraction of the concrete domains*/
choose a splitting operator ⊕_{E,i}, op[0]← i
repeat
  repeat
    e ← ⊕_{E,op[j]}(e)_{width[j]} /*backtrackable point*/
    e ← E-consistency(e)
    if τ(e) ≤ r or e ⊂ S then
      Sol[nbSol] ← e
      nbSol++
      width[j]++
    else if e = ∅ then
      width[j]++
    else
      j++
      choose a splitting operator ⊕_{E,i}, op[j]← i
    end if
  until width[j] > | ⊕_{E,op[j]} |
  j−−
  width[j]++
until width[0] > | ⊕_{E,op[0]} |
```

Fig. 2. Solving with abstract domains. The set of splitting operators is $\{\oplus_{E,i}, 1 \leq i \leq k\}$. The process stops when a precision r has been reached. Every time a choice is made on the partition obtained from the application of a splitting operator, the point is marked as backtrackable.

3.4 Solving

This section defines a generic solving process based on abstract domains and shows how to retrieve the usual continuous solving process.

Proposition 3. *If E is closed by intersection (H1), has no infinite decreasing chain (H2), and if $r \in \tau(E^f)$, then the solving process on figure 2 terminates and is complete.*

Proof. We prove here that the algorithm terminates. Assume that the search tree is infinite. Because of the definition of a splitting operator (Definition 9), its width is finite. Thus there exists an infinite branch. Along this branch, the abstract domains are strictly decreasing as long as e is not a least element of E^f. By hypothesis (H2), there exists a K such that $\forall k \geq K, e_k = e_K$. Let us study the different possible cases:

- $e_K = \emptyset$, then the algorithm terminates,
- $e_K \neq \emptyset$ and ($\tau(e_K) \leq r$ or $e_K \subset \mathcal{S}$), then the algorithm also terminates,
- $e_K \neq \emptyset$ and $\tau(e_K) > r$, then e_K is splitted and $e_K \neq e_{K+1}$ which creates a contradiction with the existence of K.

Hence the solving process on figure 2 terminates. Completeness of the solver is obvious by definition of the splitting operators. □

This proposition defines a generic solver on any abstract domain E, provided that the hypothesis (H1) and (H2) are true, and that r is well chosen. The efficiency of the solver will of course depends on the consistency algorithms on E and on the abstract representation.

The usual continuous solver can be retrieved, as shown below.

Example 6. For a fixed n, the set $\mathbb{I}^n$ with the splitting operators $\oplus_{\mathbb{I}^n,i}$ and the precision $\tau_{\mathbb{I}^n}(I) = max(\overline{I_i} - \underline{I_i})$ for $i \in \{1 \dots n\}$ is an abstract domain. In order to model the fact that the search process ends when a given precision r is reached, one can stop when $\tau_{\mathbb{I}^n} \leq r$. The $\mathbb{I}^n$ solving process corresponds to the usual continuous solvers with Hull consistency.

This example shows how to retrieve the usual continuous abstract domain, which is cartesian and of a single type (built as cartesian products of intervals).

It is now possible to define an abstract solver as the classical iteration of propagations and splits, see 2. It returns a table of abstract domains which are either smaller than r or only contains solutions ($e \subset \mathcal{S}$). The union of all these elements approximate the solution set. Notice that the choice of a splitting operator at a given depth must be stored to keep the solver completeness.

We showed that this generic solver included the usual interval solving process for continuous constraints. But it is possible to use a different domain representation. There exists several domain representation in Abstract Interpretation, here we detail a non-Cartesian representation: the Octagon.

4 Octagons

In geometry, an octagon, in $\mathbb{R}^2$, is a polygon having eight faces[2]. In this paper, we use a more general definition given in [2].

Definition 11 (Octagonal constraints). *Let v_i, v_j be two real variables. An octagonal constraint is a constraint of the form $\pm v_i \pm v_j \leq c$ with $c \in \mathbb{R}$.*

For instance in $\mathbb{R}^2$, octagonal constraints define straight lines which are parallel to the axis if $i = j$ and diagonal if $i \neq j$. The intuition is the same in $\mathbb{R}^n$, where the octagonal constraints define hyperplanes.

Definition 12 (Octagon). *Given a set of octagonal constraints $\mathcal{O}$, the subset of $\mathbb{R}^n$ points satisfying all the constraints in $\mathcal{O}$ is called an octagon.*

[2] http://mathworld.wolfram.com/Octagon.html

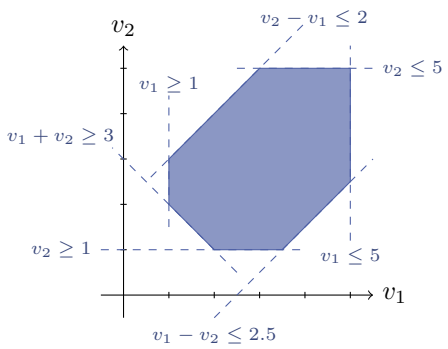


Fig. 3. Example of an octagon in $\mathbb{R}^2$. It is defined by seven non redundant octagonal constraints.

Remark 3. Here follows some general remarks on octagons:

- The geometric shape defined above includes the geometric octagons, but also other polygons (*e.g.,* in $\mathbb{R}^2$, an octagon can have less than eight faces);
- an octagon can be defined with redundant constraints (for example $v_1 - v_2 \leq c$ and $v_1 - v_2 \leq c'$), but only one of them defines a face of the octagon (the one with the lowest constant in this example),
- in $\mathbb{R}^n$, an octagon has at most $2n^2$ faces, which is the maximum number of possible non-redundant octagonal constraints on n variables,
- an octagon is a set of *real* points, but, like the intervals, they can be restricted to have floating-points bounds ($c \in \mathbb{F}$).

An example of an octagon in $\mathbb{R}^2$ is given Figure 3.

Remark 4. The octagons are closed under intersection. Let $\mathcal{O} = \{\pm v_i \pm v_j \leq c\}$ and let $\mathcal{O}' = \{\pm v_i \pm v_j \leq c'\}$ for $i, j \in \{1 \dots n\}$. Then $\mathcal{O} \cap \mathcal{O}' = \{\pm v_i \pm v_j \leq \min(c, c'), i, j \in \{1 \dots n\}\}$ is an octagon.

Remark 5. Let $\mathcal{O} = \{\pm v_i \pm v_j \leq c\}$ and let $\mathcal{O}' = \{\pm v_i \pm v_j \leq c'\}$ for $i, j \in \{1 \dots n\}$. Then $\{\mathcal{O}, \mathcal{O}'\}$ has both a *lub* $\mathcal{O} \cap \mathcal{O}'$ and a *glb* $\mathcal{O} \cup \mathcal{O}' = \{\pm v_i \pm v_j \leq \max(c, c'), i, j \in \{1 \dots n\}\}$.

Figure 4 illustrates the union and the intersection of two octagons.

In the following, octagons are restricted to floating-point octagons. Without loss of generality, we assume octagons to be defined with no redundancies.

4.1 Matrix Representation of Octagons

An octagon can be represented with a *difference bound matrix* (DBM) as described in [8, 2]. This representation is based on a normalization of the octagonal constraints as follows.

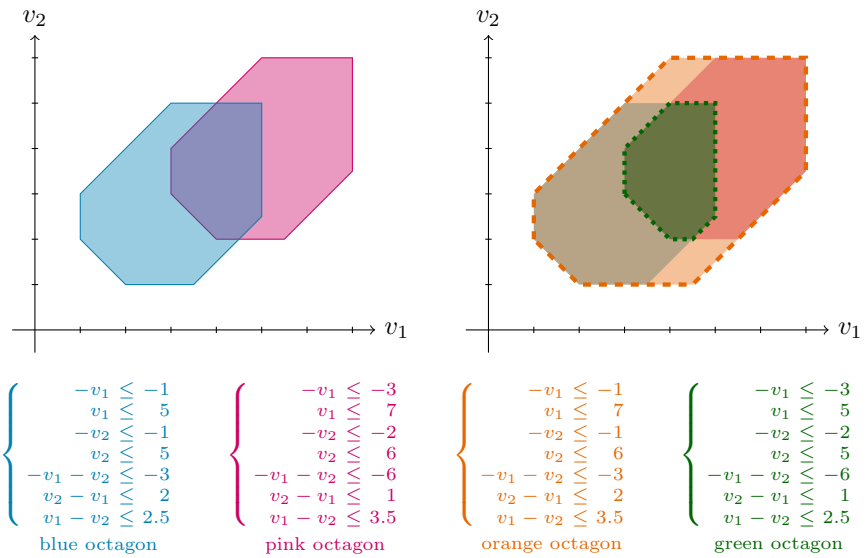
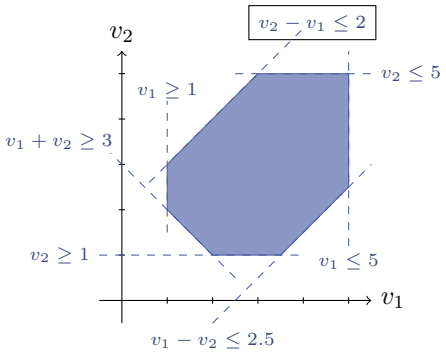


Fig. 4. Example of the union and the intersection of two octagons. The union of the blue and the pink octagons gives the orange octagon (dashed line), while the intersection gives the green one (dotted line).

Definition 13 (difference constraints). *Let w, w' be two variables. A difference constraint is a constraint of the form $w - w' \leq c$ with $c \in \mathbb{F}$ a constant.*

By introducing new variables, it is possible to rewrite an octagonal constraint as an equivalent difference constraint: let $C \equiv (\pm v_i \pm v_j \leq c)$ an octagonal constraint. Define the new variables $w_{2i-1} = v_i, w_{2i} = -v_i, w_{2j-1} = v_j$ and $w_{2j} = -v_j$. Then

- for $i \neq j$
  - if $C \equiv (v_i - v_j \leq c)$, then C is equivalent to the difference constraints $(w_{2i-1} - w_{2j-1} \leq c)$ and $(w_{2j} - w_{2i} \leq c)$,
  - if $C \equiv (v_i + v_j \leq c)$, then C is equivalent to the difference constraints $(w_{2i-1} - w_{2j} \leq c)$ and $(w_{2j-1} - w_{2i} \leq c)$,
  - if $C \equiv (-v_i - v_j \leq c)$, then C is equivalent to the difference constraints $(w_{2i} - w_{2j-1} \leq c)$ and $(w_{2j} - w_{2i-1} \leq c)$,
  - if $C \equiv (-v_i + v_j \leq c)$, then C is equivalent to the difference constraints $(w_{2i} - w_{2j} \leq c)$ and $(w_{2j-1} - w_{2i-1} \leq c)$.
- for $i = j$
  - if $C \equiv (v_i - v_i \leq c)$, then C is pointless, and can be removed,
  - if $C \equiv (v_i + v_i \leq c)$, then C is equivalent to the difference constraint $(w_{2i-1} - w_{2i} \leq c)$,
  - if $C \equiv (-v_i - v_i \leq c)$, then C is equivalent to the difference constraint $(w_{2i} - w_{2i-1} \leq c)$,



(a)

$$\begin{array}{c} \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{array} \begin{array}{c} \begin{array}{cccc} w_1 & w_2 & w_3 & w_4 \end{array} \\ \begin{pmatrix} 0 & -2 & \boxed{2} & -3 \\ 10 & 0 & +\infty & 2.5 \\ 2.5 & -3 & 0 & -2 \\ +\infty & \boxed{2} & 10 & 0 \end{pmatrix} \end{array}$$

(b)

Fig. 5. Equivalent representations for the same octagon: the octagonal constraints 5(a) and the DBM 5.

In what follows, regular variables are always written $(v_1 \dots v_n)$, and the corresponding new variables are written $(w_1, w_2, \dots w_{2n})$ with: $w_{2i-1} = v_i$, and $w_{2i} = -v_i$. As shown in [2], the rewritten difference constraints represent the same octagon as the original set of octagonal constraints, by replacing the positive and negative occurrences of the v_i variables by their w_i counterparts. Storing difference constraints is thus a suitable representation for octagons.

Definition 14 (DBM). *Let $\mathcal{O}$ be an octagon in $\mathbb{R}^n$, and its sequence of potential constraints as defined above. The octagon DBM is a $2n \times 2n$ square matrix, such that the element at row i, column j is the constant c of the potential constraint $w_j - w_i \leq c$.*

An example is shown on Figure 5(b): the element on row 1 and column 3 corresponds to the constraint $v_2 - v_1 \leq 2$ for instance.

At this stage, different DBMs can represent the same octagon. For example on Figure 5(b), the element row 2 and column 3 can be replaced by 100, for instance, without changing the corresponding octagon. In [2], an algorithm is defined so as to optimally compute the smallest values for the elements of the DBM. This algorithm is adapted from the Floyd-Warshall shortest path algorithm [9], modified in order to take advantage of the DBM structure. It exploits the fact that w_{2i-1} and w_{2i} correspond to the same variable.

We introduce a box representation for octagons. This representation, combined with the DBM will be used to define, from an initial set of continuous constraints, an equivalent system taking the octagonal domains into account.

4.2 Octagons as Intersection of Boxes

In two dimensions, an octagon can be represented by the intersection of one box in the canonical basis for $\mathbb{R}^2$, and one box in the basis obtained from the canonical basis by a rotation of angle $\pi/4$ (see figure 6(b)).

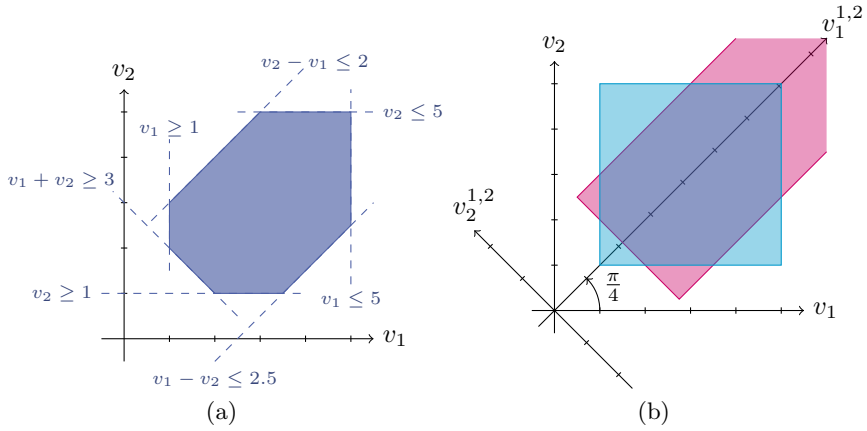


Fig. 6. Equivalent representations for the same octagon: the octagonal constraints 6(a) and the intersection of boxes 6(b).

We generalize this remark to n dimensions.

Definition 15 (Rotated basis). *Let $B = (u_1, \ldots, u_n)$ be the canonical basis of $\mathbb{R}^n$. Let $\alpha = \pi/4$. The* (i,j)-rotated basis $B_\alpha^{i,j}$ *is the basis obtained after a rotation of α in the subplane defined by (u_i, u_j), the other vectors remaining unchanged:*
$B_\alpha^{i,j} = (u_1, \ldots, u_{i-1}, (cos(\alpha)u_i + sin(\alpha)u_j), \ldots u_{j-1}, (-sin(\alpha)u_i + cos(\alpha)u_j), \ldots u_n)$.

By convention, for any $i \in \{1 \ldots n\}$, $B_\alpha^{i,i}$ represents the canonical basis. In what follows, α is always $\pi/4$ and will be omitted. Finally, every variable v living in the $B^{i,j}$ rotated basis and whose domain is D will be denoted by $v^{i,j}$ and its domain by $D^{i,j}$.

The DBM can also be interpreted as the representation of the intersection of one box in the canonical basis and $n(n-1)/2$ other boxes, each one living in a rotated basis. Let $\mathcal{O}$ be an octagon in $\mathbb{R}^n$ and its DBM M, with the same notations as above (M is a $2n \times 2n$ matrix). For $i, j \in \{1 \ldots n\}$, with $i \neq j$, let $\mathcal{B}_\mathcal{O}^{i,j}$ be the box $I_1 \times \cdots \times I_i^{i,j} \times \cdots \times I_j^{i,j} \cdots \times I_n$, in $B^{i,j}$, such that $\forall k \in \{1 \ldots n\}$

$$
\begin{aligned}
I_k &= \left[\underline{-\tfrac{1}{2} M[2k-1, 2k]},\ \overline{\tfrac{1}{2} M[2k, 2k-1]} \right] \\
I_i^{i,j} &= \left[\underline{-\tfrac{1}{\sqrt{2}} M[2j-1, 2i]},\ \overline{\tfrac{1}{\sqrt{2}} M[2j, 2i-1]} \right] \\
I_j^{i,j} &= \left[\underline{-\tfrac{1}{\sqrt{2}} M[2j-1, 2i-1]},\ \overline{\tfrac{1}{\sqrt{2}} M[2j, 2i]} \right]
\end{aligned}
$$

Proposition 4. *Let $\mathcal{O}$ be an octagon in $\mathbb{F}^n$, and $\mathcal{B}_\mathcal{O}^{i,j}$ the boxes as defined above. Then $\mathcal{O} = \bigcap_{1 \leq i,j \leq n} \mathcal{B}_\mathcal{O}^{i,j}$.*

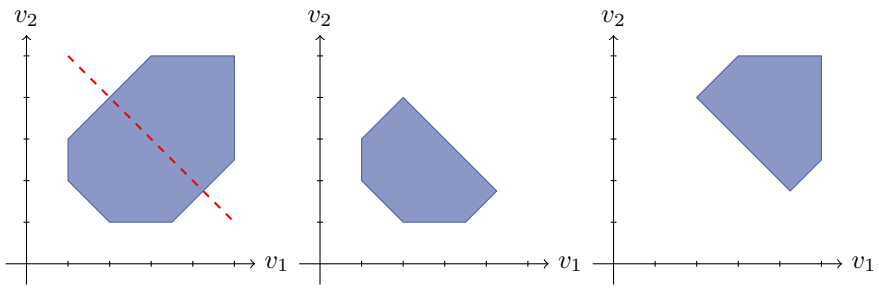


Fig. 7. Example of a split: the octagon on the left is cut in the $B^{1,2}$ basis.

Proof. Let $i, j \in \{1..n\}$. We have $v_i^{i,j} = 1/\sqrt{2}(v_i + v_j)$ and $v_j^{i,j} = 1/\sqrt{2}(v_i - v_j)$ by definition 15. Thus $(v_1 \dots v_i^{i,j} \dots v_j^{i,j} \dots v_n) \in \mathcal{B}_{\mathcal{O}}^{i,j}$ iff it satisfies the octagonal constraints on v_i and v_j, and the unary constraints for the other coordinates, in the DBM. The box $\mathcal{B}_{\mathcal{O}}^{i,j}$ is thus the solution set for these particular octagonal constraints. The points in $\bigcap_{1 \le i,j \le n} \mathcal{B}_{\mathcal{O}}^{i,j}$ are exactly the points which satisfy all the octagonal constraints. □

Example 7. Considering the DBM Figure 5(b), the boxes are $I_1 \times I_2 = [1, 5] \times [1, 5]$, and $I_1^{1,2} \times I_2^{1,2} = [3/\sqrt{2}, +\infty] \times [-2.5/\sqrt{2}, \sqrt{2}]$.

To summarize, an octagon with its DBM representation can also be interpreted as a set of octagonal constraints (definition in intension) or equivalently as an intersection of rotated boxes (definition in extension), at the cost of a multiplication / division with the appropriate rounding mode.

4.3 Octagonal Split

The octagonal split extends the usual split operator to octagons. Splits can be performed in the canonical basis, thus being equivalent to the usual splits, or in a rotated basis. It can be defined as follows:

Definition 16. *Given an octagonal domain defined with the box representation* $D_1 \dots D_n, D_1^{1,2} \dots D_1^{n-1,n} \dots D_n^{n-1,n}$, *such that* $D_k^{i,j} = [a, b]$ *with* $a, b \in \mathbb{F}$, *a splitting operator for variable* $v_k^{i,j}$, *computes the two octagonal subdomains* $D_1 \dots [a, \overline{(a+b)/2}] \dots D_n^{n-1,n}$ *and* $D_1 \dots [\underline{(a+b)/2}, b] \dots D_n^{n-1,n}$.

As for the usual split, the union of the two octagonal subdomains is the original octagon, thus the split does not lose solutions. This definition does not take into account the correlation between the variables of the different basis. We take advantage again of the octagonal representation to communicate the domain reduction to the other basis. A split is thus immediately followed by a Floyd-Warshall propagation. Figure 7 shows an example of the split.

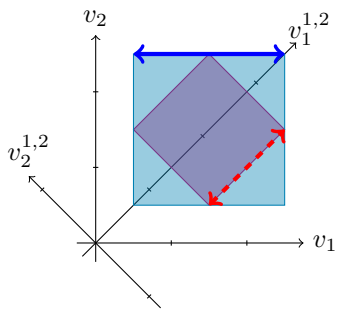


Fig. 8. Example of the precision: the usual continuous precision returns the the size of D_1 (the blue plain double arrow) while the octagonal precision returns the size of $D_1^{1,2}$ (the red dashed double arrow).

4.4 Precision

In most continuous solvers, the precision is defined as the size of the largest domain. For octagons, this definition leads to a loss of information because it takes the domains separately and does not take into account the correlation between the variables (*e.g.*, $v_1^{\{1,2\}}$ and $v_2^{\{1,2\}}$ depend on v_1 and v_2).

Definition 17. *Let $\mathcal{O}$ be an octagon, and $I_1 \dots I_n, I_1^{1,2} \dots I_n^{n-1,n}$ its box representation. The octagonal precision is $\tau(\mathcal{O}) = min_{1 \leq i,j \leq n}(max_{1 \leq k \leq n}(\overline{I_k^{i,j}} - \underline{I_k^{i,j}}))$.*

For a single regular box, τ would be the same precision as usual. On an octagon, we take the minimum precision of the boxes in all the bases because it is more accurate, and it allows us to retrieve the operational semantics of the precision, as shown by the following proposition: in an octagon of precision r overapproximating a solution set $\mathcal{S}$, every point is at a distance at most r from $\mathcal{S}$.

Proposition 5. *Let $< v_1 \dots v_n, D_1 \dots D_n, C_1 \dots C_p >$ be a CSP, and $\mathcal{O}$ an octagon overapproximating $\mathcal{S}$. Let $r = \tau(\mathcal{O})$. Let $(v_1, \dots v_n) \in \mathbb{R}^n$ be a point of $D_1 \times \dots \times D_n$. Then $\forall 1 \leq i \leq n$, $min_{s \in \mathcal{S}} |v_i - s_i| \leq r$, where $s = (s_1 \dots s_n)$. Each coordinate of all the points in $\mathcal{O}$ are at a distance at most r of a solution.*

Proof. By definition 17, the precision r is the minimum of some quantities in all the rotated basis. Let $B^{i,j}$ be the basis that realizes this minimum. Because the box $\mathcal{B}^{i,j} = D_1 \times \dots \times D_i^{i,j} \times \dots D_j^{i,j} \dots \times D_n$ is Hull-consistent by proposition 9, it contains $\mathcal{S}$. Let $s \in \mathcal{S}$. Because $r = max_k(\overline{D_k} - \underline{D_k})$, $\forall 1 \leq k \leq n$, $|s_k - v_k| \leq \overline{D_k} - \underline{D_k} \leq r$. □

Figure 8 shows an example of the precision. While the usual precision returns the size of D_1, the octagonal precision returns the size of $D_1^{1,2}$.

4.5 Octagonal Abstract Domain

Proposition 6. *The set of octagons forms a complete lattice* $\mathbb{O}$.

Proof. Any set of octagons has both a *lub* and a *glb* (remark 4 and remark 5). Thus $\mathbb{O}$ is a complete lattice.

Proposition 7. *There exists a Galois connection* $\mathbb{O} \rightleftarrows \mathbb{B}$

Proof.
- $\alpha : \mathbb{O} \to \mathbb{B}, \mathcal{O} \mapsto \{\pm v_i \leq c, i \in \{1 \dots n\}\}$ with $\mathcal{O} = \{\pm v_i \pm v_j \leq c\}$: only the constraints of the form $v_i \leq c$ are kept which correspond to the box bounds.
- $\gamma : \mathbb{B} \to \mathbb{O}, \mathcal{B} \mapsto \mathcal{B}$: is straightforward and exact as a box is an octagon. □

From Proposition 6 and Proposition 7, we can say that octagons can be the base set of an abstract domain.

With this definition of the octagons we create all the possible basis ($\frac{n(n+1)}{2}$). Among all these basis some are maybe less relevant than others. We present in the following subsection a partial definition of the octagons.

4.6 Partial Octagon Abstract Domain

As said previously, when an octagon is created, a large number of basis are created. This transforms a CSP with n variables into a representation with n^2 variables. Among all the created basis maybe some are less interesting w.r.t. the problem to solve. We thus define a Partial Octagon abstract domain as follow:

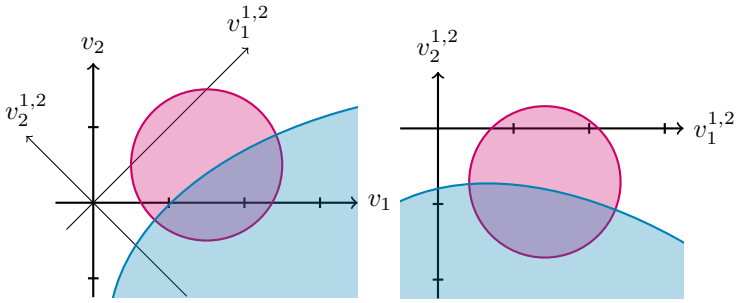
Definition 18 (Partial Octagon). *Let* $\mathcal{J} \in \mathcal{P}(\{1 \dots n\})$ *a set of indexes. Let* $\mathcal{O}$ *be a set of octagonal constraints* $\{\pm v_i \pm v_j,\ i, j \in \mathcal{J}\}$. *The subset of* $\mathbb{R}^n$ *points satisfying all the constraints in* $\mathcal{O}$ *is called a partial octagon.*

The definition remains the same, it is still a set of points satisfying a set of octagonal constraints, only that not all the possible non redundant octagonal constraints are defined. Note that this definition includes the octagons (when $\mathcal{J} = \{1 \dots n\}$). It can easily be shown that the properties on the octagons hold for the partial octagons.

In the following sections, we show how the octagons can be implemented in a solver.

5 Octagonal Solving

In order to have an octagonal solver, we first need to transform a CSP into an octagonal CSP. Then define an octagonal consistency and a propagation scheme. The first three subsections describe these three steps and subsection 5.4 gives some properties of the octagonal solver. In order to guide the solving process, one can define different variable heuristics, some are presented subsection 5.5. The last subsection presents the different heuristics used to generate partial octagons given a CSP.



(a) The constraints in the original basis.

(b) The constraints in the $(1,2)$-rotated basis.

Fig. 9. Example of rotated constraints: comparison between the original CSP 9(a) and the rotated CSP 9(b).

5.1 Octagonalization of a CSP

Consider a CSP on variables $(v_1 \dots v_n)$ in $\mathbb{R}^n$. The goal is now to equip this CSP with an octagonal domain. We detail here how to build an octagonal CSP from a regular one, and show that the two systems are equivalent.

First, the CSP is associated to an octagon, by stating all the possible octagonal constraints $\pm v_i \pm v_j \leq c_k$ for $i, j \in \{1 \dots n\}$. The constants c_k represent the bounds of the octagon boxes and are dynamically modified. They are initialized to $+\infty$.

The rotations defined in the previous section introduce new axes, that is, new variables $(v_i^{i,j}, v_j^{i,j}$ in the (i,j)-rotated basis). Because these variables are redundant with the regular ones, they are also linked through the CSP constraints $(C_1 \dots C_p)$, and these constraints have to be rotated as well.

Given a function f on variables $(v_1 \dots v_n)$ in B, the expression of f in the (i,j)-rotated basis is obtained by symbolically replacing the i-th and j-th coordinates by their expressions in $B_\alpha^{i,j}$. Precisely, replace v_i by $\cos(\alpha)v_i^{i,j} - \sin(\alpha)v_j^{i,j}$ and v_j by $\sin(\alpha)v_i^{i,j} + \cos(\alpha)v_j^{i,j}$ where $v_i^{i,j}$ and $v_j^{i,j}$ are the coordinates for v_i and v_j in $B_\alpha^{i,j}$. The other variables are unchanged.

Definition 19 (Rotated constraint). *Given a constraint C_k holding on variables $(v_1 \dots v_n)$, the (i,j)-rotated constraint $C_k^{i,j}$ is the constraint obtained by replacing each occurrence of v_i by $\cos(\alpha)v_i^{i,j} - \sin(\alpha)v_j^{i,j}$ and each occurrence of v_j by $\sin(\alpha)v_i^{i,j} + \cos(\alpha)v_j^{i,j}$.*

Figure 9 compares the constraints in the original basis to the constraints in the $(1,2)$-rotated basis.

Given a continuous CSP $< v_1 \dots v_n, D_1 \dots D_n, C_1 \dots C_p >$, we define an octagonal CSP by adding the rotated variables, the rotated constraints, and the rotated domains stored as a DBM. To sum up and fix the notations, the octagonal CSP thus contains:

- the regular variables $(v_1 \dots v_n)$;
- the rotated variables $(v_1^{1,2}, v_2^{1,2}, v_1^{1,3}, v_3^{1,3} \dots v_n^{n-1,n})$, where $v_i^{i,j}$ is the i-th variable in the (i,j)-rotated basis $B_\alpha^{i,j}$;
- the regular constraints $(C_1 \dots C_p)$;
- the rotated constraints $(C_1^{1,2}, C_1^{1,3} \dots C_1^{n-1,n} \dots C_p^{n-1,n})$.
- the regular domains $(D_1 \dots D_n)$;
- a DBM which represents the rotated domains. It is initialized with the bounds of the regular domains for the cells at position $2i, 2i-1$ and $2i-1, 2i$ for $i \in \{1 \dots n\}$, and $+\infty$ everywhere else.

In these conditions, the initial CSP is equivalent to this transformed CSP, restricted to the variables $v_1 \dots v_n$, as shown in the following proposition.

Proposition 8. *Consider a CSP* $< v_1 \dots v_n, D_1 \dots D_n, C_1 \dots C_p >$, *and the corresponding octagonal CSP as defined above. The solution set of the original CSP* $\mathcal{S}$ *is equal to the solution set of the* $(v_1 \dots v_n)$ *variables of the octagonal CSP.*

Proof. Let $s \in \mathbb{R}^n$ a solution of the octogonal CSP for $(v_1 \dots v_n)$. Then $s \in D_1 \times \dots \times D_n$ and $C_1(s) \dots C_p(s)$ are all true. Hence s is a solution for the original CSP. Reciprocally, let $s' \in \mathbb{R}^n$ a solution of the original CSP. The regular constraints $(C_1 \dots C_p)$ are true for s'. Let us show that there exist values for the rotated variables such that the rotated constraints are true for s'. Let $i, j \in \{1 \dots n\}$, $i \neq j$, and $k \in \{1 \dots p\}$ and $C_k^{i,j}$ the corresponding rotated constraint. By definition 19, $C_k^{i,j}(v_1 \dots v_{i-1}, \cos(\alpha)v_i^{i,j} - \sin(\alpha)v_j^{i,j}, v_{i+1} \dots \sin(\alpha)v_i^{i,j} + \cos(\alpha)v_j^{i,j} \dots v_n) \equiv C_k(v_1 \dots v_n)$. Let us define the two reals $s_i^{i,j} = cos(\alpha)s_i + sin(\alpha)s_j$ and $s_j^{i,j} = -sin(\alpha)s_i + cos(\alpha)s_j$, the image of s_i and s_j by the rotation of angle α. By reversing the rotation, $cos(\alpha)s_i^{i,j} - sin(\alpha)s_j^{i,j} = s_i$ and $sin(\alpha)s_i^{i,j} + cos(\alpha)s_j^{i,j} = s_j$, thus $C_k^{i,j}(s_1 \dots s_i^{i,j}, \dots s_j^{i,j} \dots s_n) = C_k(s_1 \dots s_n)$ is true. It remains to check that $(s_1 \dots s_i^{i,j}, \dots s_j^{i,j} \dots s_n)$ is in the rotated domain, which is true because the DBM is initialized at $+\infty$. □

For a CSP on n variables, this representation has an order of magnitude n^2, with n^2 variables and domains, and at most $p\frac{n(n-1)}{2} + p$ constraints. Of course, many of these objects are redundant. We explain in the next sections how to use this redundancy to speed up the solving process.

As said in section 3.3, one need to define given an abstract domain, the propagators as they depend both on the constraint and the abstract domain. The next sections show how the consistency and the propagation scheme can be computed for the octagons.

We first generalize the Hull-consistency definition to the octagonal domains, and define propagators for the rotated constraints. Then, we use the modified version of Floyd-Warshall (as described in section 4.1) to define an efficient propagation scheme for both octagonal and rotated constraints.

5.2 Octagonal Consistency

We generalize the definition of Hull-consistency on intervals for any continuous constraint to the octagons. With the box representation, we show that any propagator for Hull-consistency on boxes can be extended to a propagator on the octagons. For a given n-ary relation on $\mathbb{R}^n$, there is a unique smallest octagon (w.r.t. inclusion) which contains the solutions of this relation, as shown in the following proposition.

Remark 6. Consider a constraint C (resp. a constraint sequence $(C_1 \dots C_p)$), and $\mathcal{S}_C$ its set of solutions (resp. $\mathcal{S}$). Then there exist a unique octagon $\mathcal{O}$ such that: $\mathcal{S}_C \subset \mathcal{O}$ (resp. $\mathcal{S} \subset \mathcal{O}$), and for all octagons $\mathcal{O}'$, $\mathcal{S}_C \subset \mathcal{O}'$ implies $\mathcal{O} \subset \mathcal{O}'$. $\mathcal{O}$ is the unique smallest octagon containing the solutions, w.r.t. inclusion.

Proposition 9. *Let C be a constraint, and $C^{i,j}$ the (i,j)-rotated constraint for $i,j \in \{1 \dots n\}$. Let $\mathcal{B}^{i,j}$ be the Hull-consistent box for $C^{i,j}$, and $\mathcal{B}$ the Hull-consistent box for C. The Oct-consistent octagon for C is the intersection of all the $\mathcal{B}^{i,j}$ and $\mathcal{B}$.*

5.3 Propagation Scheme

The propagation scheme presented in subsection 4.1 for the octagonal constraints is optimal. We thus rely on this propagation scheme, and integrate the non-octagonal constraints propagators in this loop. The point is to use the octagonal constraints to benefit from the relational properties of the octagon. This can be done thanks to the following remark: all the propagators defined in the previous subsections are monotonic and complete (as is the HC4 algorithm). It results that they can be combined in any order in order to achieve consistency, as shown for instance in [6].

The key idea for the propagation scheme is to interleave the refined Floyd-Warshall algorithm and the constraint propagators. A pseudocode is given on figure 10. At the first level, the DBM is recursively visited so that the minimal bounds for the rotated domains are computed. Each time a DBM cell is modified, the corresponding propagators are added to the propagation list. The propagation list is applied before each new round in the DBM (so that a cell that would be modified twice is propagated only once). The propagation is thus guided by the additional information of the relational domain. This is illustrated on Figure 11: the propagators $\rho_{C_1} \dots \rho_{C_p}, \rho_{C_1^{1,2}} \dots \rho_{C_p^{n-1,n}}$ are first applied (11(a), 11(b)), then the boxes are made consistent w.r.t. each other using the refined Floyd-Warshall algorithm (11(c)).

We show here that the propagation as defined on figure 10 computes the consistent octagon for a sequence of constraints.

The refined Floyd-Warshall has a time complexity of $O(n^3)$. For each round in its loop, in the worst case we add p propagators in the propagation list. Thus the time complexity for the propagation scheme of figure 10 is $O(n^3 p^3)$. In the end, the octagonal propagation uses both representations of octagons. It

float dbm[2n, 2n] */*the dbm containing the octagonal constraints*/*
list propagList $\leftarrow (\rho_{C_1}, \ldots \rho_{C_p}, \rho_{C_1^{1,2}} \ldots \rho_{C_p^{n-1,n}})$ */*list of the propagators to apply*/*
while propagList $\neq \emptyset$ **do**
  apply all the propagators of propagList */*constraints propagation*/*
  propagList $\leftarrow \emptyset$
  for i,j **from** 1 **to** n **do**
    m $\leftarrow$ minimum of (dbm$[2i-1, k]$+dbm$[k, 2j-1]$) for k from 1 to $2n$
    m $\leftarrow$ minimum(m, (dbm$[2i-1, 2i]$+dbm$[2j, 2j-1]$)/2)
    if m $<$ dbm$[2i-1, 2j-1]$ **then**
      dbm$[2i-1, 2j-1] \leftarrow$ m */*update of the DBM*/*
      add $\rho_{C_1^{i,j}} \ldots \rho_{C_p^{i,j}}$ **to** propagList */*get the propagators to apply*/*
    end if
    repeat the 5 previous steps **for** dbm$[2i-1, 2j]$, dbm$[2i, 2j-1]$, **and** dbm$[2i, 2j]$
  end for
end while

Fig. 10. Pseudo code for the propagation loop mixing the Floyd Warshall algorithm (the *for* loop) and the regular and rotated propagators $\rho_{C_1} \ldots \rho_{C_p}, \rho_{C_1^{1,2}} \ldots \rho_{C_p^{n-1,n}}$, for an octagonal CSP as defined in section 5.1.

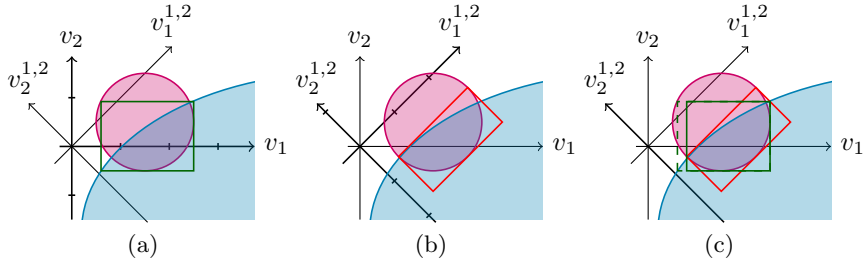


Fig. 11. Example of the Oct-consistency: an usual consistency algorithm is applied in each basis (Figures 11(a) and 11(b)) then the different boxes are made consistent using the modified Floyd-Warshall algorithm (Figure 11(c)).

takes advantage of the relational property of the octagonal constraints (Floyd-Warshall), and of the usual constraint propagation on boxes (propagators). This comes to the cost of computing the octagon, but is expected to give a better precision in the end.

Besides the expected gain in precision obtained with octagon consistency, the box representation of octagons allows us to go a step further and define a fully octagonal solver.

5.4 Octagonal Solver

We showed section 4.5 that an abstract domain can be based on the octagons, and with the consistency and the propagation scheme described previously, we thus obtain an octagonal solver. The solving process is the same as in Figure

2 using the octagonal abstract domain. As octagons are closed by intersection (H1) and $\mathbb{O}$ has no infinite decreasing chain (H2), this solving process terminates. And by Figure 2 it returns a sequence of octagons whose union overapproximate the solution space. Precisely, it returns either octagons for which all points are solutions, or octagons overapproximating solution sets with a precision r.

To guide the exploration of the search space one can define a variable heuristic. Different heuristics are presented in the following subsection.

5.5 Variable Heuristics

An important feature of a constraint solver is the variable heuristic. It chooses which variable to split. For continuous constraints, one usually chooses to split the variable that has the largest domain. This heuristic aims at reaching faster the precision by cutting the largest domain in half. The three following heuristics rely on this strategy the only difference is the set of variables that can be splited. Let V' be the set of all variables in the octagonal CSP $V' = (v_1 \dots v_n, v_1^{1,2}, v_2^{1,2} \dots v_{n-1}^{n-1,n}, v_n^{n-1,n}) = (v_1 \dots v_n, v_{n+1} \dots v_{n^2})$.

LargestFirst (LF) As the usual continuous split, the variable to split is chosen among all the variables. The variable to split is the variable v_i which realizes the maximum of

$$\arg\max_{i \in \{1 \dots n^2\}} (\overline{D_i} - \underline{D_i})$$

LargestCanFirst (LCF) Splits only the regular variables, which are the variables of the original CSP. The variable to split is the variable v_i which realizes the maximum of

$$\arg\max_{i \in \{1 \dots n\}} (\overline{D_i} - \underline{D_i})$$

LargestOctFirst (LOF) Splits only the rotated variables, which are the variables generated by the rotations. The variable to split is the variable v_i which realizes the maximum of

$$\arg\max_{i \in \{n+1 \dots n^2\}} (\overline{D_i} - \underline{D_i})$$

Those three strategies do not take into account the correlation between the variables. And the variable which has the largest domain can be in a basis that is of little interest for the problem (it probably has a wide range because the constraints are poorly propagated in this basis). We thus define an octagonal strategy.

Oct-Split (OS) This strategy relies on the same remark as for definition 17: the variable to split is the variable $v_k^{i,j}$ which realizes the minimum of

$$\min_{1 \le i,j \le n} (\max_{1 \le k \le n} (\overline{D_k^{i,j}} - \underline{D_k^{i,j}}))$$

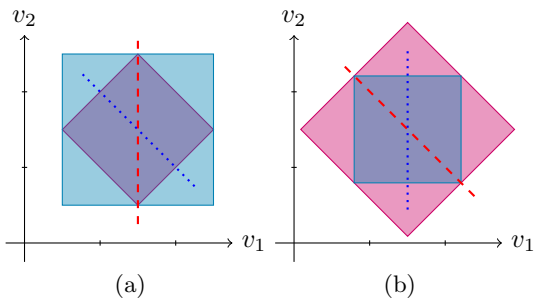


Fig. 12. Examples of the different variable heuristics. Figure 12(a), the LF and LCF strategies split along the red dashed line while the LOF and OS strategies split along the blue dotted line. Figure 12(b), the LF and LOF strategies split along the red dashed line while the LCF and OS strategies split along the blue dotted line.

The strategy is the following: choose first a promising basis, that is, a basis in which the boxes are tight (choose i, j). Then take the worst variable in this basis as usual (choose k). This heuristic aims at reaching faster the octagonal precision.

Figure 12 shows examples of the different variables heuristics.

5.6 Octagonalisation heuristic

In previous works, all the $n(n+1)/2$ possible basis were generated, which for problems of great dimension (large value for n) created an octagon with a large number of faces. As said previously, maybe some of the generated basis are less interesting than others w.r.t. the problem to solve. We thus define different strategies to determine a subset of bases to generate. Given a problem we try to find, by symbolically looking at the constraints, which bases are more likely to improve the search. In other words, we try to determine in which bases the problem is easier to solve by looking at the constraints. This step is done once at the creation of the octagon not like in [10] where the basis changes at each step of the resolution.

ConstraintBased The basis $B_\alpha^{i,j}$ is generated only if the variables v_i and v_j appear in the same constraint. If v_i and v_j appear in the constraint C, then C links v_i with v_j. We say that there exists a relation between v_i and v_j. The idea behind this heuristic is to emphasize a relation that already exists between two variables. In the worst case, all the pairs of variables appear in the constraints thus all the basis will be generated.

Random Among all the possible bases generates *one* random basis. Chooses i and j with $i < j$ then generates the basis $B_\alpha^{i,j}$. The number of generated bases relies on a tradeoff between precision and computation times, the more bases

are generated the longer it takes to solve the problem but the more precise the solutions are because the octagons have more sides. As we try to reduce the computation time, we set the number of generated bases to one, this is purely arbitrary and can be random for a pure random heuristic.

StrongestLink Generates only one basis, the one for which the variables have the strongest link. In other words, the basis corresponding to the pair of variables which appears in most constraints. The idea is to emphasize the strongest relation between two variables. We want the new variables to be the most constrained as possible so that the solving process will have more chances to reduce their domains.

Example 8. Given the following constraints:

$$\begin{cases} v_1 + v_2 + v_1 \times v_2 \leq 3 \\ \cos(v_1) + v_3 \leq 10 \\ v_1 \times v_3 \geq 1 \end{cases}$$

The basis $B_\alpha^{1,3}$ is generated as v_1 and v_3 appear together in two constraints while the pair $\{v_1, v_2\}$ appears in only one constraint.

Promising Generates the basis $B_\alpha^{i,j}$ maximizing the number of promising patterns ($\pm v_i \pm v_j$ or $\pm v_i \times v_j$) in the constraints. These patterns can be easily simplified in the rotated basis. For instance, let $a = \cos(\pi/4) = \sin(\pi/4)$, $v_i + v_j = av_i^{i,j} - av_j^{i,j} + av_i^{i,j} + av_j^{i,j}$ is equal to $2av_i^{i,j}$ and $v_i \times v_j = (av_i^{i,j} - av_j^{i,j})(av_i^{i,j} + av_j^{i,j})$ is equal to $(av_i^{i,j})^2 - (av_j^{i,j})^2$. With this heuristic, we want to reduce the number of multiple occurrences of the new variables in the rotated constraints. By reducing the number of multiple occurrences, the consistency will be more efficient and thus, the solving process should be faster.

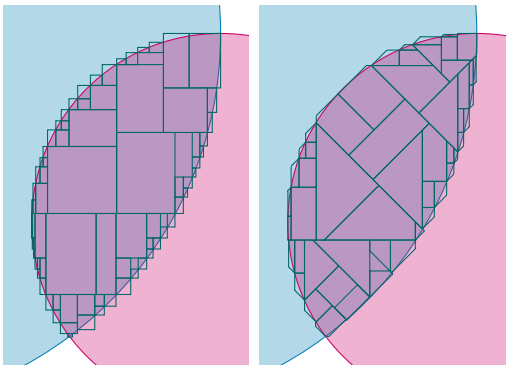
Example 9. Given the following constraints:

$$\begin{cases} v_1 + v_2 + v_1 \times v_2 \leq 3 \\ \cos(v_1) + v_3 \leq 10 \\ v_1 \times v_3 \geq 1 \end{cases}$$

The basis $B_\alpha^{1,2}$ is generated as there are two promising patterns with v_1 and v_2 ($v_1 + v_2$ and $v_1 \times v_2$) while for the pair $\{v_1, v_3\}$ there is only one promising pattern ($v_1 \times v_3$).

6 Experiments

This section compares the octagonal solver with a traditional interval solver on classical benchmarks.



(a) (b)

Fig. 13. Comparison of the results obtained with the interval resolution 13(a) to the results obtained with the octagonal resolution 13(b), given a time limit of 10ms.

6.1 Implementation

We have implemented a prototype of the octagonal solver, with Ibex, a C++ library for continuous constraints [11]. We use the Ibex implementation of *HC4-Revise* [12] to contract the constraints. The octagons are implemented with their DBM representation. Additional rotated variables and constraints are posted and dealt with as explained above.

An important point is the rotation of the constraints. The HC4 algorithm is sensitive to multiple occurrences of the variables, and the symbolic rewriting defined in section 5.1 creates multiple occurrences. Thus, the HC4 propagation on the rotated constraints could be very poor if performed directly on the rotated constraints. It is necessary to simplify the rotated constraints w.r.t. the number of multiple occurrences for the variables. We use the Simplify function of Mathematica to do this. The computation time indicated below does not include the time for this treatment, however, it is negligible compared to the solving times. The propagator is an input of our method: we used a standard one (HC4), but more recent propagator such as [13] will be considered in the future. It is sufficient for our needs that the consistency algorithms computes overapproximations of the Hull-consistent boxes, as it is often the case for continuous propagators. Further work include adding propagators such as [13], which better takes into account the symbolic expression of the constraint to improve the propagation.

6.2 Methodology

We have tested the prototype octagonal solver on problems from the Coconut benchmark[3]. These problems have been chosen depending on the type of the constraints (inequations, equations, or both) and on the computation time needed

[3] This benchmark can be found at http://www.mat.univie.ac.at/ neum/glopt/coconut/

name	# var	ctr type	LF	LCF	LOF	OS
brent-10	10	=	-	362.77	-	**337.48**
o32	5	$\leq$	104.62	**40.60**	122.82	40.74
ipp	8	=	5 105.17	282.319	5 787.5	**279.36**
trigo1	10	=	-	256.71	-	**253.53**
KinematicPair	2	$\leq$	**59.72**	62.91	60.74	60.78
nbody5.1	6	=	105.93	27.33	121.33	**27.08**
bellido_P	9	=	375.81	327.74	-	**318.09**
pramanik	3	=	396.23	**240.93**	414.76	240.96
caprasse_P	4	=	2902.76	2387.52	-	**2353.58**
h74	4	= $\leq$	2896.58	1553.67	4036.89	**647.76**

Table 1. Comparison of the different variable heuristics on problems from the Coconut benchmark. The first three columns give the name of the problem, the number of variable and the type of the constraints. For each problem, we give the CPU time in seconds to solve it using the LargestFirst heuristic (LF), the LargestCanFirst (LCF), the LargestOctFirst (LOF) ant the Oct-Split (OS).

to solve using the usual continuous solving. We took only the problems for which the interval solving process took more than 20 seconds.

Experiments have been made, with Ibex 1.18, on a MacBook Pro Intel Core 2 Duo 2.53 GHz. Apart from the variable heuristic presented in subsection 5.5, the experiments have been done with the same configuration in Ibex, in particular, using the same propagators, so as to compare exactly the octagonal results with their interval counterparts. Also we set the precision parameter r to 0.001 and a time limit of 3 hours in both cases.

6.3 Results

Figure 13 compares the results obtained with the interval resolution to the results obtained with the octagonal solver given a time limit of 10ms. Table 1 compares for each selected problem the results obtained in terms of time for the different variable heuristics. Table 2 compares for each selected problem the time (in seconds) obtained by the intervals to those obtained by the octagons and the partial octagons. Table 3 compares for each selected problem the number of boxes created during the interval solving process to the number of octagons created during the octagonal solving. Finally, Table 4 compares for each selected problem the number of boxes solution to the number of octagons solution. In all the tables, the dash symbol '-' stands for 'time out'.

6.4 Analysis

Table 1 compares the results of the different variable heuristics. We can see that the Oct-Split strategy is most of the time the best in term of time. By splitting in the basis with good interest, it efficiently explores the search space. It also suit the octagonal precision which is part of the termination criterion.

name	$\mathbb{I}^n$	$\mathbb{O}$	CB	R	SL	P
brent-10	**21.58**	330.73	89.78	92.59	105.91	109.74
o32	27.25	40.74	40.74	**17.63**	20.68	21.23
ipp	38.83	279.36	279.36	30.14	**29.07**	41.60
trigo1	40.23	253.53	253.53	38.60	**37.03**	**37.03**
KinematicPair	**59.04**	60.78	60.78	60.78	60.78	60.78
nbody5.1	95.99	27.08	**22.13**	443.07	50.87	439.69
bellido	**111.12**	-	-	362.69	361.56	318.09
pramanik	281.80	240.96	240.96	141.94	137.42	**131.30**
caprasse	9175.36	-	-	**1085.21**	1131.33	2353.58
h74	-	647.76	647.76	-	**0.15**	**0.15**

Table 2. Results on problems from the Coconut benchmark. The first column gives the name of the problem. For each problem, we give the CPU time in seconds to solve it using the intervals ($\mathbb{I}^n$), the octagons ($\mathbb{O}$) and all the partial octagons, Constraint Based (CB), Random (R), Strongest Link (SL) and Promising (P).

name	$\mathbb{I}^n$	$\mathbb{O}$	CB	R	SL	P
brent-10	211 885	**5 467**	5 841	175 771	205 485	21 495
o32	161 549	**25 319**	**25 319**	52 071	62 911	72 565
ipp	237 445	**21 963**	**21 963**	51 379	50 417	75 285
trigo1	13 621	**4 425**	**4 425**	6 393	5 943	5 943
KinematicPair	847 643	**373 449**	**373 449**	**373 449**	**373 449**	**373 449**
nbody5.1	598 521	5 435	**5 429**	578 289	137 047	542 263
bellido	774 333	-	-	577 367	573 481	**496 729**
pramanik	1 992 743	**243 951**	**243 951**	346 633	315 861	319 037
caprasse	150 519 891	-	-	**4 445 655**	4 472 839	9 933 597
h74	-	418 867	418 867	-	**625**	**625**

Table 3. Results on problems from the Coconut benchmark. The first column gives the name of the problem. For each problem, we give the number of box/octagons created while solving with the intervals ($\mathbb{I}^n$), the octagons ($\mathbb{O}$) and all the partial octagons, Constraint Based (CB), Random (R), Strongest Link (SL) and Promising (P).

For these reasons we used this heuristic while comparing the octagonal solving to the interval resolution.

Looking only at the time, Table 2, both octagons and partial octagons obtained very bad results for problems `brent-10` and `bellido`. In these two problems, for each rotated basis only few constraints are rotated. If few constraints are rotated then the rotated variables have less chances to be reduced and thus do not help reducing the original variables. This does not add enough new information and we lost time propagating the rotated constraints. On the contrary, for problems `nbody5.1`, `caprasse` and `h74` the octagonal resolution or one of the partial octagonal resolution really improves the results. We were expected those results given the form of the constraints ($v_i - v_j \leq c$ in `h74` for instance). For the rest of the problems, the time obtained by the partial octagonal resolution is quite similar to the one obtained by the interval resolution.

name	$\mathbb{I}^n$	$\mathbb{O}$	CB	R	SL	P
brent-10	825	**149**	153	636	1 643	1 765
o32	74 264	**12 523**	**12 523**	35 868	31 216	25 833
ipp	2 301	**2243**	**2243**	4 329	6 922	6 194
trigo1	40	**32**	**32**	168	140	140
KinematicPair	346 590	**186 717**	**186 717**	**186 717**	**186 717**	**186 717**
nbody5.1	1 012	1 003	**996**	1 794	50 526	2 230
bellido	**7 372**	-	-	34 756	32 482	37 508
pramanik	149 011	**54 659**	**54 659**	72 052	69 621	71 239
caprasse	1 544	-	-	164	148	**110**
h74	-	209 406	209 406	-	**293**	**293**

Table 4. Results on problems from the Coconut benchmark. The first column gives the name of the problem, the number of variable and the type of the constraints. For each problem, we give the number of box/octagons in the computed solution by the intervals ($\mathbb{I}^n$), the octagons ($\mathbb{O}$) and all the partial octagons, Constraint Based (CB), Random (R), Strongest Link (SL) and Promising (P).

Still looking at the time, we can see that the Strongest Link octagonalization (SL) is better than the Promising one (P). The more rotated constraints there are in a rotated basis, the better is the propagation.

Now, looking at Table 3, we can see that the number of octagons created during the resolution is always less than the number of boxes created. As the octagons are more precise than the boxes, we created less octagons but spend more time computing at each node of the search.

7 Related Works

Our work is related to [2], in static analysis of programs. Their goal is to compute overapproximations for the traces of a program. The octagons are shown to provide a good trade off between the precision of the approximation and the computation cost. We use their matrix representation and we adapted their version of the Floyd-Warshall algorithm.

Propagation algorithms for the difference constraints, also called *temporal constraints*, have already presented in [14, 15]. They have a better complexity than the one we use, but are not suited to the DBM case, because they do not take into account the doubled variables.

The idea of rotating variables and constraints has already been proposed in [10], in order to better approximate the solution set. The resolution process is still the same, the propagators are applied and if the resulting consistent box does not meet some termination criterion, a splitting operator is applied. But when a box is splitted, new rotated basis are computed for each of the new boxes. Another difference between our method and theirs is that their method is dedicated to under-constrained systems of equations.

8 Conclusion

In this paper, we have proposed a generic solving algorithm for continuous constraints and illustrated it with the octagonal case. Since domains in Constraint Programming can be interpreted as components of a global multidimensional parallelepipedic domain, we have constructed octagonal approximations on the same model and provided algorithms for octagonal CSP transformations, filtering, propagation, precision and splitting. An implementation based on Ibex and preliminary experimental results on classical benchmarks are encouraging, particularly in the case of systems containing inequalities. Future work involves the experimental study of other interval-based propagators such as Mohc [13] and extensions to other geometric structures such as interval polyhedron [16].

References

1. P. Cousot, R. Cousot, in *Proceedings of the 2nd International Symposium on Programming* (1976), pp. 106–130
2. A. Miné, Higher-Order and Symbolic Computation **19**(1), 31 (2006)
3. F. Rossi, P. van Beek, T. Walsh, *Handbook of Constraint Programming (Foundations of Artificial Intelligence)* (Elsevier, 2006)
4. D. Goldberg, ACM Computing Surveys **23**(1), 5 (1991)
5. K.R. Apt, Theoretical Computer Science **221** (1999)
6. F. Benhamou, in *Proceedings of the 5th International Conference on Algebraic and Logic Programming* (1996), pp. 62–76
7. A. Miné, Domaines numériques abstraits faiblement relationnels. Ph.D. thesis, École Normale Supérieure (2004)
8. M. Menasche, B. Berthomieu, in *Protocol Specification, Testing, and Verification* (1983)
9. R. Floyd, Communications of the ACM **5**(6) (1962)
10. A. Goldsztejn, L. Granvilliers, Constraints **15**(2), 190 (2010). DOI http://dx.doi.org/10.1007/s10601-009-9082-3
11. G. Chabert, L. Jaulin, Artificial Intelligence **173**, 1079 (2009)
12. F. Benhamou, F. Goualard, L. Granvilliers, J.F. Puget, in *Proceedings of the 16th International Conference on Logic Programming* (1999), pp. 230–244
13. I. Araya, G. Trombettoni, B. Neveu, in *Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI 2010* (2010)
14. R. Dechter, I. Meiri, J. Pearl, in *Proceedings of the first international conference on Principles of Knowledge Representation and Reasoning* (1989)
15. J.C. Régin, M. Rueher, *Inequality-sum : a global constraint capturing the objective function* (RAIRO Operations Research, 2005)
16. L. Chen, A. Miné, J. Wang, P. Cousot, in *Proceedings of the 16th International Static Analysis Symposium (SAS'09)* (2009), pp. 309–325