From Implicit to Explicit Pavings

Gilles Chabert, Rémi Douence

From Implicit to Explicit Pavings

Gilles Chabert, Rémi Douence

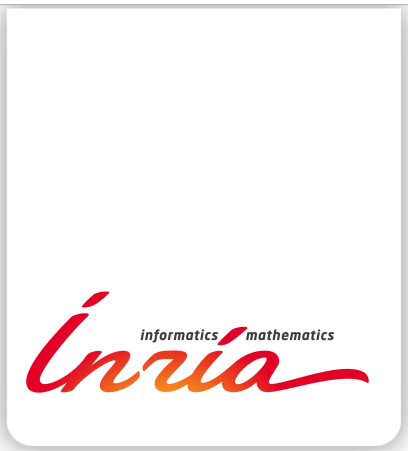
RESEARCH REPORT

N° 8028

June 2012

Project-Teams Ascola and Tasc

ISSN 0249-6399 ISRN INRIA/RR--8028--FR+ENG



From Implicit to Explicit Pavings

Gilles Chabert, Rémi Douence

Project-Teams Ascola and Tasc

Research Report n° 8028 — June 2012 — 17 pages

Abstract: A combinatorial search can either be performed by using an implicit search tree, where an initial state is recursively transformed until some goal state is reached, or by using an explicit search tree, where an initial tree structure containing the root state is iteratively expanded until the leaves match the set of goal states. This paper proposes an exploratory study aimed at showing that explicit search trees can play a distinguished role in the field of numerical constraints. The first advantage of an explicit search is expressiveness: we can write new algorithms or reformulate existing ones in a simple and unified way. The second advantage is efficiency, since an implicit search may also lead to a blowup of redundant computations. This is illustrated through various examples.

Key-words: Constraints, continuous domain, paving

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

From Implicit to Explicit Pavings

Résumé : Une recherche combinatoire peut être réalisée soit en utilisant un arbre de recherche implicite, où l'état initial est récursivement transformé jusqu'à atteindre un état but, soit en utilisant un arbre de recherche explicite, où la structure d'arbre initiale contenant l'état racine est itérativement étendue jusqu'aux feuilles qui satisfont l'ensemble des états buts. Ce rapport propose une étude exploratoire visant à montrer que les arbres de recherche explicites peuvent jouer un rôle notable dans le cas des contraintes numériques. Le premier avantage de la recherche explicite est son expressivité : on peut écrire de nouveaux algorithmes ou reformuler des algorithmes existants d'une manière simple et unifiée. Le second avantage est l'efficacité car une recherche explicite permet d'éviter, dans certains cas, une explosion de calculs redondants. Ceci est illustré à travers plusieurs exemples.

Mots-clés : Contraintes, domaines continus, pavage

1 Introduction

A combinatorial search, especially in the field of constraint programming, is always a node-centered algorithm where a *local* transformation is applied to the state represented by the node. Basically, this transformation consists in a filtering operation followed by a choice point (variable instantiation or bisection in the case of continuous domains).

A semantically equivalent algorithm consists in applying a *global* transformation to the set of all nodes. This global transformation selects one or several nodes and apply the local transformations to each, resulting in a new search tree. This tree is transformed in turn and the process is repeated until no transformation are possible. Therefore, instead of a recursive algorithm, one gets an iterative algorithm.

There is usually no point in using explicit search trees for combinatorial problems. As just said, algorithms are equivalent while managing the tree structure introduces an overhead in time and memory.

In contrast, explicit search trees have a clear advantage in numerical constraint programming, merely because they represent *pavings* in this context, a meaningful concept for both users and people that design algorithms.

Let us focus first on the user side. With continuous domains, the output is not a collection of leaves that satisfy a goal state. This is an important difference with discrete constraint programming, where one is looking for one or all the solutions. It is rather a disjunction of Euclidean sets that partition the initial space (such a partition forms what we call a paving). The mathematical definition of the output, i.e., the semantic of the solver, relates pavings, not individual tuples.

This point has already been argued in [2]. However, the algorithms proposed therein build pavings only implicitly. The explicit representation is only accessible to the user as a top-level result.

We go a step further in this paper by showing that explicit pavings is also the right structure for the programmers. Indeed, in many cases, the final sets are obtained after a sequence of operations on intermediate sets until some fixpoint is reached. The explicit representation of pavings therefore allows to program algorithms as they are really thought. The first advantage is a considerable simplification over casting these algorithms into an implicit search. The second is the ability to write new algorithms or strategies.

We illustrate our purpose through various examples. We also give a new and simple formalism for writing algorithms based on explicit pavings. This formal setting is first defined in Section 2. Examples are introduced in the subsequent sections.

1.1 Notations and background

A box is a vector of intervals. Boxes are denoted with brackets, like $[x]$. The smallest box enclosing a set $\mathcal{S}$ is $\Box\mathcal{S}$. A *constraint* is a subset of reals, most of the time described implicitly by a relation between the variables. Typically, a relation is an equation, an inequality.

A *contractor* is an operator C that takes a box $[x]$ as input and return a box $C([x])$ such that $C([x]) \subseteq [x]$. We shall not need to consider additional properties.

Given a constraint c, we call *inner* (resp. *outer*) contractor for c, a contractor C that satisfies:

$$\begin{array}{lll} \forall [x] & x \in \Big([x] \backslash C([x])\Big) \Longrightarrow c(x) & \text{(inner contractor)} \\ \forall [x] & x \in [x] \wedge c(x) \Longrightarrow x \in C([x]) & \text{(outer contractor)} \end{array} \tag{1}$$

We call *trace* the pair $(C([x]), [x] \backslash C([x]))$. This pair represents a contraction by the remaining and removed part of the contracted box.

We assume all throughout this paper that inner and outer contractors can be built for regular constraints and, in particular, for logical combinations of inequalities. These contractors are atomic operators of our language.

We shall also consider the *precision contractor* $P\langle\varepsilon\rangle$ introduced in [2] (the notation will be justified soon). Given a real value $\varepsilon > 0$, the contractor $P\langle\varepsilon\rangle$ contracts a box $[x]$ to the empty set if its width (some measure of its size) is less than ε (in short, we will say that $[x]$ is a ε-box), and leaves it intact otherwise:

$$P\langle\varepsilon\rangle([x]) = \begin{cases} \emptyset & \text{if } \text{width}([x]) < \varepsilon, \\ [x] & \text{otherwise} \end{cases}$$

2 The language

For the sake of clarity, algorithms in this paper are written in functional style.

If f is a function and x an argument, $(f\ x)$ is the application of f to x. For instance, if C is a contractor and $[x]$ a box, $(C\ [x])$ is the contracted box. The definition of a function may also involve parameters. We will use angle brackets to specify parameters. For instance $P\langle\varepsilon\rangle$ is the precision contractor with a precision set to ε. A parameter is always fixed. Finally, types will be denoted with the standard arrow-based notation from type theory. E.g., the signature of a function that takes a contractor and return a box is: Box→ Box→ Box.

We deliberately remove all the minor technical details of the programs to keep the essence of algorithms (the full code can be freely downloaded from the authors' web pages). Algorithms essentially rely on three different data types and a couple of operators on them. Algorithms are also all instances of a very simple pattern. This setting defines our *language*. We present it now.

Data types.

There are three data types: *box*, *subpaving* and *paving*:

1. a *box* is a cross product of intervals. The type name is Box. For the sake of simplicity, this type does not take account of the dimension.

2. a *subpaving* is a list of equidimensional boxes. The power-set union of the boxes represent a subset of $\mathbb{R}^n$, where n is the common dimension. Blackboard letters like $\mathbb{K}$ are kept for subpavings. The type name is Subp.

3. a *paving* is a list of subpavings. Contrary to boxes and subpaving, the list does not represent a set but a collection of sets. For this reason, pavings will be always handled explicitly as fixed-sized tuples like $(\mathbb{K}_1, \ldots, \mathbb{K}_k)$ thanks to pattern matching. The type of a paving is written (Subp,...,Subp).

Basic operations.

Usual set-theoretical operations are allowed on boxes and subpavings: *intersection, union, projection, set difference, cross product and projection.* Their semantic is straightforward. We will use for convenience the mathematical notation, e.g., $[x] \cup [y]$ is the union of two boxes. The

projection of a box onto a subset of variables x is denoted by π_x. Let us just point out that the complementary of a box is not a box but a subpaving.
Of course, any operation on boxes or subpavings can be introduced in the language. Besides the basic ones above, we will need further the following ones:

- **bisect**::Box→ Subp.
  Split a box $[x]$ into two subboxes $[x]_1, [x_2]$ and return the subpaving $[x]_1 \cup [x_2]$.
- **trace**$\langle C \rangle$::Box→ (Subp,Subp), where C is a contractor.
  Return the trace of C on a box $[x]$, that is $(C([x]), [x] \backslash C([x]))$.
- **pick**::Box→ Box.
  Pick randomly a point inside the box and return it (as a degenerated box).
- **min**$\langle y \rangle$::Subp→ Box, where y is the name of a variable.
  (**min**$\langle y \rangle$ $\mathbb{K}$) returns a box of $\mathbb{K}$ that minimizes the upper bound of the component named y.

Complex operations.

Our language has just one complex operation, the *map*, that comes in three variants, one for each data type.

- **map**::(Box→ Box)→ Subp→ Subp
  Given f that maps a box to a box, (**map** f) takes a subpaving $\mathbb{K} = [x]_1 \cup \ldots \cup [x]_n)$ as argument, applies f to each box of $\mathbb{K}$ and collect the resulting boxes into a new subpaving:

  $$\textbf{map}\ f\ ([x]_1 \cup \ldots \cup [x]_n) := (f\ [x]_1])\ \cup\ \ldots\ \cup\ (f\ [x]_n) \tag{2}$$

  Consider for instance a contractor C and a subpaving $\mathbb{X}$. (**map** $C\,\mathbb{X}$) returns a subpaving $\mathbb{X}'$ where each box of $\mathbb{X}$ has been contracted with C.

- **map***::(Box→ Subp)→ Subp→ Subp
  Same definition as **map**, except that f maps a box to a subpaving. The union on the right side of (2) is performed between subpavings instead of plain boxes. For instance, the expression

  $$\textbf{map}^*\ \text{bisect}\ \mathbb{X}$$

  bisects each box of $\mathbb{X}$ in two subboxes (a subpaving) and all these pairs of subboxes are flattened in a single subpaving. The result describes mathematically the same set as $\mathbb{X}$ but with twice the number of boxes.

- **map****::(Box→ (Subp,...,Subp))→ Box→ (Subp,...,Subp)
  Same definition as **map***, except that f maps a box to a paving. The union on the right side of (2) is performed elementwise.

  For instance, consider the expression

  $$\textbf{map}^{**}\ \text{trace}\langle P\langle\varepsilon\rangle\rangle\ \mathbb{X}\,.$$

  For each box $[x]$, the **trace** operator applied with $P\langle\varepsilon\rangle$ returns either $([x], \emptyset)$ or $(\emptyset, [x])$ depending on whether $[x]$ is an ε-box or not respectively. All theses traces are collected and gathered elementwise leading to a paving $(\mathbb{E}, \mathbb{X}')$ where $\mathbb{E}$ is the set of all the ε-boxes of $\mathbb{X}$ and $\mathbb{X}'$ the other ones.

The fixpoint pattern.

As said in introduction, algorithms are fixpoint on subpavings. This means that the result is reached when some condition

$$\mathbb{K} = (f\ \mathbb{K})$$

holds for a subpaving $\mathbb{K}$ and a function f :: SUBP→SUBP.

However, checking that two subpavings are equal is not practical. It is much easier to check that a subpaving is empty and, clearly, a fixpoint can be restated as a code that iterates until the *delta* between the current result and the expected one vanishes. For this reason, the pattern of algorithms will be the following one:

Algorithm 1: algo :: SUBP→ SUBP

```
1 algo 𝕏 =;
2 if (𝕏 = ∅) then ∅ else
3     let  (𝕂, 𝕏′)=map** f 𝕏;
4     in 𝕂 ∪ (algo 𝕏′)                    // the keyword in means "return"
```

- $\mathbb{X}$ represents the "delta" subpaving, that is, what is left to be processed.
- f :: BOX→ (SUBP,SUBP) is a function that, given a box, returns two subpavings. Intuitively, the first subpaving represents the contribution of the box to the result. This contribution usually involves only a *part* of the box so that the second subpaving corresponds to the unprocessed part (or residu).

We will also resort to a variant with accumulator:

Algorithm 2: algo :: (SUBP,SUBP) → SUBP

```
1 algo (𝕏, 𝕂)=;
2 if (𝕏 = ∅) then 𝕂 else
3     let  (𝕂′, 𝕏′)=map** f 𝕏;
4     in algo (𝕏′, 𝕂∪𝕂′)
```

The body of the **let** will be broken into multiple lines below, for readability.

First illustration.

Let us illustrate the language right away on a simple search.

The following program has to enclose all the points that satisfy a constraint c into ε-sized boxes.

The subpaving $\mathbb{X}$ contains the set of current –unprocessed– boxes (or, equivalently, the set of pending nodes in the search tree). The subpaving $\mathbb{E}$ contains the solutions found so far, that is, ε-sized boxes that could not be filtered out (or, equivalently, the leaves of the search tree). Comments follow.

The first step consists in contracting all the boxes of $\mathbb{X}$, by *mapping* the outer contractor C w.r.t the constraint. This results in an intermediate subpaving $\mathbb{X}_1$. The second step separates from $\mathbb{X}_1$ the boxes that are too small (put in $\mathbb{E}'$) from the others (put in $\mathbb{X}_2$). The third step bisects all the remaining boxes, those in $\mathbb{X}_2$ which yields $\mathbb{X}_3$. Finally, the algorithm is called again on $\mathbb{X}_3$, with the new set of solutions: the union of $\mathbb{E}$ and the ones found during this call, $\mathbb{E}'$.

Algorithm 3: search :: (SUBP,SUBP) → SUBP

```
search (𝕏,𝔼)=;
if (𝕏 = ∅) then 𝔼 else
    let 𝕏₁ = map C 𝕏                                   // contract all the boxes;
        (𝔼′,𝕏₂) = map** trace⟨P⟨ε⟩⟩ 𝕏₁      // remove those that are small enough;
        𝕏₃ = map* bisect 𝕏₂                              // bisect all the boxes;
    in search (𝕏₃, 𝔼 ∪ 𝔼′)
```

3 Set Inverse and variants

Since the apparition of the SIVIA algorithm [9], computing the *set inverse* has been a classical problem in continuous constraint programming [1, 11, 16]. Given a mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ and a box $[y] \subset \mathbb{R}^m$, the question is to find a paving that characterizes the set $\mathcal{S}_1$ of all preimages of $y \in [y]$:

$$\mathcal{S}_1 := \{x \in \mathbb{R}^n, \quad f(x) \in [y]\}.$$

Usually, this paving is a pair of subpavings $(\overline{\mathbb{K}}, \mathbb{E})$ where $\overline{\mathbb{K}}$ and $\mathbb{E}$ correspond respectively to the *inner* and *boundary* approximation of $f^{-1}([y])$, that is:

$$\overline{\mathbb{K}} \subseteq \mathcal{S}_1 \text{ and } \mathcal{S}_1 \subseteq \mathbb{E} \cup \overline{\mathbb{K}}.$$

First, $\mathcal{S}_1$ can be easily redefined as the set of x that satisfy a conjunction of inequalities [2] and we said in introduction that an inner contractor $\overline{C}$ and an outer contractor C can be easily built with respect to such a constraint.

It is then easy to program the set inversion in our language. Remind that $P\langle\varepsilon\rangle$ is the contractor in charge of removing small boxes on the boundary. Notice that we have parametrized **setinv** with C, $\overline{C}$ and ε, as we will call this function further for calculating different set inverses or with a different precision.

Algorithm 4: **setinv**$\langle C, \overline{C}, \varepsilon\rangle$ :: SUBP→(SUBP,SUBP,SUBP)

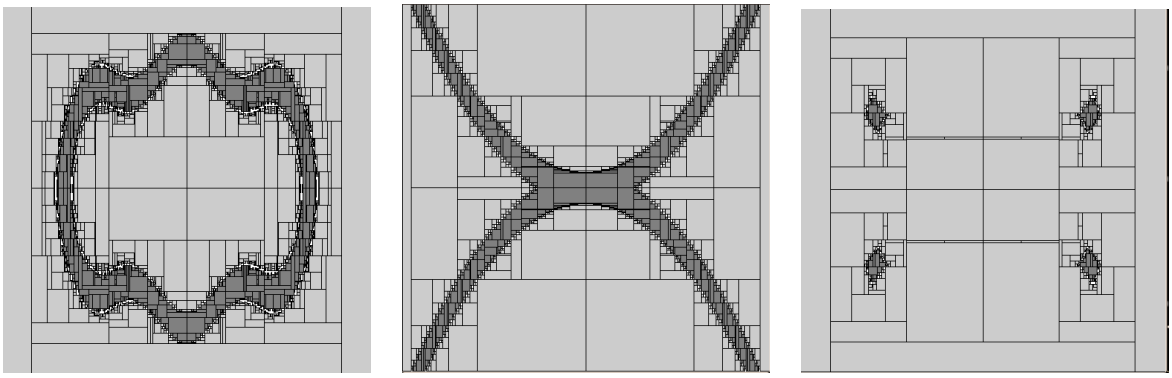
```
setinv⟨C, C̄, ε⟩ 𝕏=;
if (𝕏 = ∅) then (∅,∅,∅) else
    let (𝕂̄,𝕏₁) = map** trace⟨C̄⟩ 𝕏          // contract all the boxes w.r.t.  C̄;
        (𝕂,𝕏₂) = map** trace⟨C⟩ 𝕏₁          // contract all the boxes w.r.t.  C;
        (𝔼,𝕏₃) = map** trace⟨P⟨ε⟩⟩ 𝕏₂    // contract all the boxes w.r.t.  P⟨ε⟩;
        𝕏₄ = map* bisect 𝕏₃                              // bisect all the boxes;
        (𝕂₁,𝕂̄₁,𝔼₁)=setinv⟨C, C̄, ε⟩ 𝕏₄  in (𝕂 ∪ 𝕂₁, 𝕂̄ ∪ 𝕂̄₁, 𝔼 ∪ 𝔼₁)
```

We show now that this version based on explicit pavings allow different usage that leads to interesting variants of the set inversion problem.

Assume first that a paving of a box $[x]$ has to be calculated with respect to the intersection $\mathcal{S}_1 \cap \mathcal{S}_2$ of two sets. $\mathcal{S}_1$ (see Figure 1.(a)) is associated with an outer contractor C_1 and an inner contractor $\overline{C}_1$. Similarly, $\mathcal{S}_2$ (see Figure 1.(b)) is associated with C_2 and $\overline{C}_2$. A first alternative is simply to combine the contractors as in **main1** [2]. Figure 1.(c) shows the result obtained.

Algorithm 5: **main1** :: →(SUBP,SUBP)

```
setinv⟨C₁ ∩ C₂, C̄₁ ∪ C̄₂, ε⟩ [x]
```



(a) **(b)** **(c)**

Figure 1: **Set inversion**. On each picture, the interior of a set is described in dark gray, the exterior in light gray. **(a).** The set $\mathcal{S}_1$. The interior $\overline{\mathbb{K}}_1$ and the boundary $\mathbb{E}_1$ (white boxes that are hardly visible) are the two intermediate sets calculated in **main2**. **(b).** The set $\mathcal{S}_2$. **(c).** $\mathcal{S}_1 \cap \mathcal{S}_2$ as returned by **main1**.

Let us assume now that $\mathcal{S}_1$ is an ubiquitous constraint. A second strategy consists in preprocessing $\mathcal{S}_1$, in order to avoid duplicated computations. For that end, we first run a paver with $(C_1, \overline{C}_1)$ and then a second paver with $(C_2, \overline{C}_2)$ fed with the result paving of the first paver. In the next code, $\mathcal{S}_1$ is represented by $\overline{\mathbb{K}}_1$ and $\mathbb{E}_1$. By restricting the domain of the variable to $\overline{\mathbb{K}}_1$, the membership to the first set is de facto satisfied. Only membership to $\mathcal{S}_2$ has to be checked. This leads to the program **main2**. The difference with **main1** is emphasized in Figure 2.

Algorithm 6: **main2** :: $\rightarrow$(SUBP,SUBP)

```
1 let  (−, K̄1, E1)=setinv⟨C1, C̄1, ε⟩ [x];
2      (−, K̄, E)=setinv⟨C2, C̄2, ε⟩ K̄1;
3      (−, E′, E″)=setinv⟨C2, C̄2, ε⟩ E1  in (K̄, E ∪ E′ ∪ E″)
```

This strategy, however, has the inconvenient that all the small boxes of $\overline{\mathbb{K}}_1 \cup \mathbb{E}_1$ have to be proceeded systematically, giving no chance to C_2 or $\overline{C}_2$ to perform a global filtering on some potentially large subspaces of the initial box (that is, to fail more quickly). We can therefore imagine a compromise between entirely preprocessing $\mathcal{S}_1$ and only applying contractors dynamically since both can be potentially disadvantageous. So the third variant is to set a coarser precision ε' in the first paver and to use the resulting paving $\overline{\mathbb{K}}_1 \cup \mathbb{E}_1$ only as a rough description of $\mathcal{S}_1$. The second paver is run with a better precision $\varepsilon < \varepsilon'$ and the description of $\mathcal{S}_1$ is refined where necessary: only ε'-boxes have to be contracted again with respect to $\mathcal{S}_1$. This variant can be obtained by combining the precision contractor with the input contractors (in line with examples of [2]). This corresponds to the program **main3**. The process is depicted in Figure 3.

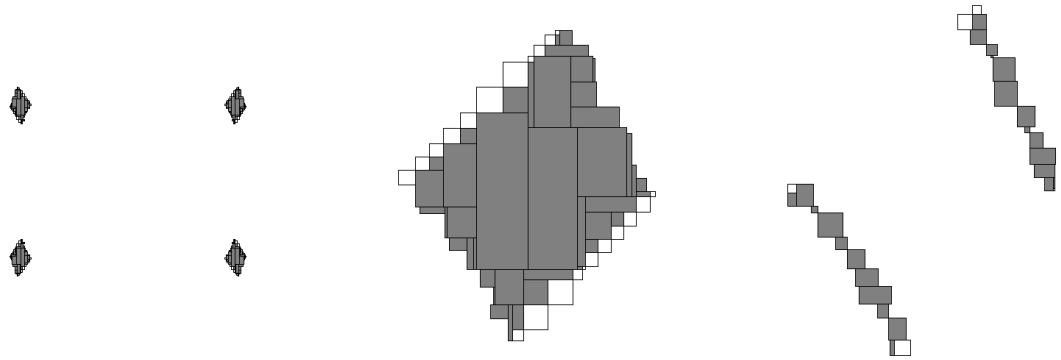


Figure 2: . **Two-step set inversion. (a).** A global view of $\overline{\mathbb{K}}$ and $\mathbb{E}$. **(b).** A zoom on the upper-right corner of (a). $\overline{\mathbb{K}}$ is in dark gray and $\mathbb{E}$ in white. $\mathbb{E}$ is inside $\overline{\mathbb{K}}_1 \subseteq \mathcal{S}_1$ and in the boundary of $\mathcal{S}_2$. If we compare with Figure 1.(b), we can check that the white boxes only surround $\overline{\mathbb{K}}$ at the boundary of $\mathcal{S}_2$. **(c).** Same zoom on the sets $\mathbb{E}'$ (dark gray) and $\mathbb{E}''$ resulting from the set inversion of $\mathbb{E}_1$.

Algorithm 7: **main3**

1 **let** $(-, \overline{\mathbb{K}}_1, \mathbb{E}_1)$=**setinv**$\langle C_1, \overline{C}_1, \varepsilon' \rangle$ $[x]$;
2 $(-, \overline{\mathbb{K}}, \mathbb{E})$=**setinv**$\langle C_2 \cap (C_{\varepsilon'} \cup C_1), \overline{C}_2, \varepsilon \rangle$ $\overline{\mathbb{K}}_1$;
3 $(-, \overline{\mathbb{K}}', \mathbb{E}')$=**setinv**$\langle C_2 \cap (C_{\varepsilon'} \cup C_1), \overline{C}_2 \cup C_{\varepsilon'} \cup \overline{C}_1, \varepsilon \rangle$ $\mathbb{E}_1$ **in** $(\overline{\mathbb{K}} \cup \overline{\mathbb{K}'}, \mathbb{E} \cup \mathbb{E}')$

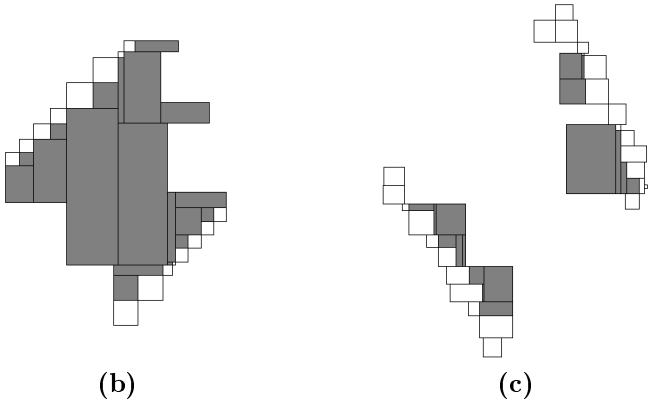


Figure 3: **Compromise between static and dynamic paving**. **(b).** A zoom on $\overline{\mathbb{K}}$ (in dark gray) and $\mathbb{E}$ (in white). We see that $\overline{\mathbb{K}}$ is smaller than in Figure 2.(b) as the input subpaving $\overline{\mathbb{K}}_1$ describes the interior of $\mathcal{S}_1$ less accurately. **(c).** Same zoom on $\overline{\mathbb{K}}'$ and $\mathbb{E}'$. The missing part of the inner subpaving ($\overline{\mathbb{K}}'$) is extracted from the coarse boundary of $\mathcal{S}_1$ ($\mathbb{E}_1$) by applying again $\overline{C}_1$ on the smaller boxes.

4 Global Optimization

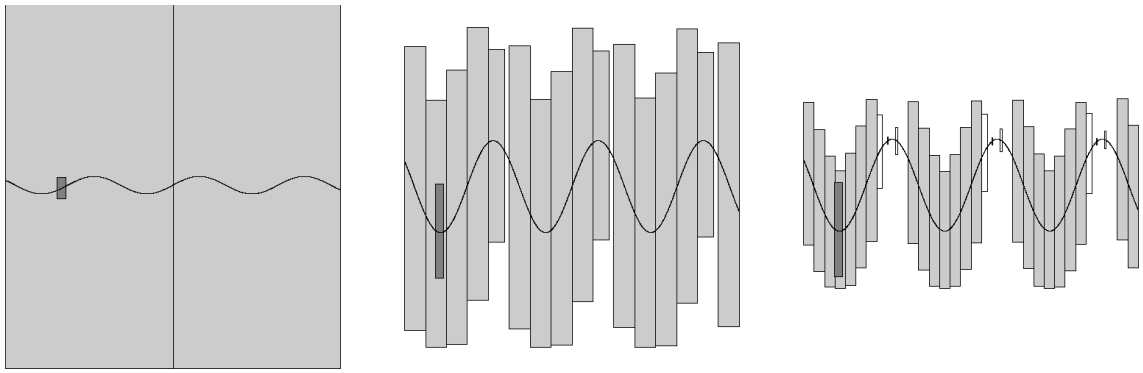
Global optimization [6, 12, 15] is to find a pair (x, y) such that $y = f(x)$ and $f(x)$ is the minimum of f over an initial domain $[x]^0$, that is, $\forall x' \in [x]^0$, $f(x') \geq f(x)$. One of the fundamental reasons that make global optimization algorithms and solvers different is that optimization requires to manage *global* information on the criterion to be minimized; this information is shared by all the nodes of the search so that we can filter with the most up-to-date bound on the criterion.

With explicit pavings, there is no such *local* versus *global* information. We can simply iterate in a fixpoint loop with a subpaving that contains the current domain for x and a box that contains the "best pair" (x_o, y_o) found so far, with $y_o = f(x_o)$. The best pair minimizes f over the part we have already explored (and removed) from the initial domain. Both structures are at the same level. We do not present a competitive algorithm here. We just show that the globality of the bounding process with respect to the criterion can be naturally integrated.

The algorithm requires an outer contractor C_f for the constraint $f(x) \leq y$. The initial call is (**optimize** $[x]^0$ $\emptyset$ $[x]^0 \times (-\infty, +\infty)$). Comments follow.

Algorithm 8: **optimize** :: (SUBP,SUBP,BOX) $\rightarrow$ SUBP

```
1 optimize (𝕏, 𝔼, opt) = ;
2 if 𝕏 = ∅ then 𝔼 else
3     let 𝕆=map ((id,f) ∘ pick) 𝕏  // ((id,f) x) returns (x, f(x))     [x'_o]×[y'_o]=min⟨y⟩
      (𝕆 ∪ [x_o]×[y_o]);
4         𝕏_1=map C_f (𝕏 ×[y'_o]);
5         𝔼_1=map C_f (𝔼 ×[y'_o]);
6         (𝔼_2, 𝕏_2)=map** trace⟨P⟨ε⟩⟩ 𝕏_1;
7         𝕏_3 = map* bisect 𝕏_2  in optimize (𝕏_3, 𝔼_1 ∪ 𝔼_2, [x'_o] × [y'_o])
```



(a) **(b)** **(c)**

Figure 4: **Global optimization.** At each step, the function f is superimposed to the paving. $\mathbb{X}$ is in light gray, $\mathbb{E}$ in white and **opt** in dark gray. **(a).** Nothing could be filtered and **opt** is not the best possible. The unique box of $\mathbb{X}$ has been bisected, $\mathbb{E} = \emptyset$.**(b).** Boxes of $\mathbb{X}$ that cross the upper part of f are contracted and **opt** encompasses a global minimizer. **(c).** ε-boxes appear, which are not discarded because the range of f on these boxes is close to the current minimum.

The algorithm is illustrated in Figure 4. $\mathbb{X}$ is the current subpaving, $\mathbb{E}$ the small boxes and **opt**

the box $[x_o]\times[y_o]$ that contains the best minimizer found so far. **Line 3.** We pick a pair $(x, f(x))$ inside each box of $\mathbb{X}$. We get a subpaving $\mathbb{O}$ of such pairs. **Line 3.** The new best candidate is the pair $[x'_o]\times[y'_o]$ that minimizes y among all the candidate pairs, which include those in $\mathbb{O}$ and the current best pair. We can filter now the boxes with our new bound on the criterion, y'_o. **Line 5.** Same thing for the ε-boxes. **Lines 6-7.** As before, we separate the small boxes from the others, which are bisected. The algorithm is called again

5 Set Image

The set image problem [8] is somehow dual to the set inverse. The purpose is to calculate a paving that describes the image $\mathcal{S}$ of a function over a set $\mathcal{S}_x$ that is itself described by a constraint (typically, a conjunction of inequalities):

$$S := \{y \in \mathbb{R}^m, \quad \exists x \in \mathcal{S}_x \subset \mathbb{R}^n,\ y = f(x)\}.$$

We reformulate the main loop of an algorithm proposed by Goldsztejn & Jaulin in [5]. First of all, they give in this paper a function that given a sufficiently small box $[x]$ and $[y] \supseteq range(f,[x])$, returns true iff $[y] \subseteq range(f, \mathcal{S}_x)$.

It is therefore easy to derive from that algorithm an operator Φ ::Box→(Subp, Subp) that either returns a pair $([y],\emptyset)$ with $[y]$ satisfying both $range(f,[x]) \subseteq [y]$ and $[y] \subseteq range(f,\mathcal{S}_x)$, or the pair $(\emptyset,[x])$.

Their main loop can now be cast into our framework. The algorithm **setimage** uses also an outer contractor C for the relation $x \in \mathcal{S}_x$. The top level call is (**setimage** $(-\infty,\infty)^n$ $\emptyset$ $\emptyset$). Comments follow.

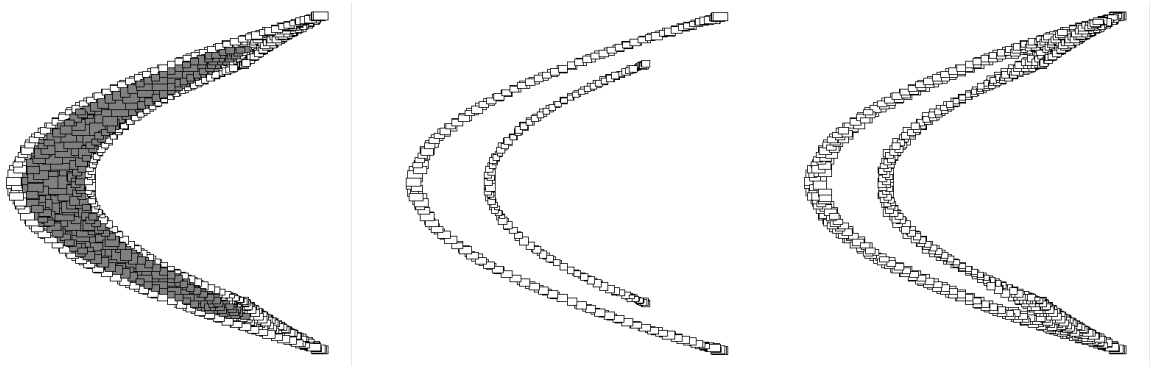
Algorithm 9: **setimage** :: (Subp,Subp,Subp) → (Subp,Subp)

```
1 setimage (𝕏, 𝕐, 𝔼_y) = ;
2 if 𝕏 = ∅ then (𝕐, 𝔼_y) else
3    let  𝕏_1 = map C 𝕏;
4         (𝕏_2, 𝕐_1) = map** Φ 𝕏_1;
5         (𝔼_x, 𝕏_3) = map** trace⟨P⟨ε⟩⟩ 𝕏_2;
6         𝔼_1 = map f 𝔼_x                    // if a box x is ε-sized, put f(x) in 𝔼_1;
7         𝕏_4 = map* bisect 𝕏_3  in setimage (𝕏_4, (𝕐 ∪ 𝕐_1), (𝔼_y ∪ 𝔼_1)))
```

$\mathbb{Y}$ and $\mathbb{E}_y$ represent respectively the current inner and boundary subpaving of $\mathcal{S}$. **Line 3.** We first filter the current subpaving w.r.t. the constraint $x \in \mathcal{S}_x$. **Line 4.** We apply Φ on each box and collect the results on two subpavings: $\mathbb{Y}_1$ that contains inner boxes of $\mathcal{S}$ and $\mathbb{X}_2$, the residual x-boxes. **Line 5.** We extract from the latter set the small boxes, $\mathbb{E}_x$. Basically, either these boxes are at the border of $\mathcal{S}_x$ or their images by f are at the border of $\mathcal{S}$. In both cases, the images of these boxes have to be included in the boundary subpaving of $\mathcal{S}$ (see Figure 5). So we apply f at Line 6 and get a subpaving $\mathbb{E}_1$ that has to be merged with $\mathbb{E}_y$.

6 Invariant set

Given a mapping f from $\mathbb{R}^n$ to $\mathbb{R}^n$, the problem can be stated as finding the largest subset S of S^0 that maps to itself, that is:



(a) **(b)** **(c)**

Figure 5: **Set image. (a).** The result obtained on the example of [5]. **(b).** The image by f of the boundary of $\mathcal{S}_x$ (obtained via set inversion). $\mathcal{S}_x$ is a ring delimited by two circles, so there are two connected components, each being the image of a circle. **(c).** The set $\mathbb{E}_y$, that is, the boundary of $\mathcal{S}$. We can see that it encompasses the subpaving of (b) and also contains extra boxes.

$$f(S) = S \quad \wedge \quad \forall S',\ S \subset S' \subseteq S^0, \quad f(S') \neq S'.$$

This problem is relevant in the context of dynamical systems, e.g., for characterizing the basin of capture [10].

We propose a simple strategy for addressing this problem. The result supplied by the next algorithm is a set of ε-boxes that forms an outer representation of S. The idea behind is somehow to apply the following iteration: $\mathbb{X} \leftarrow f(\mathbb{X})$. Of course, to make $f(\mathbb{X})$ closer to the actual range of f over $\mathbb{X}$, we need to bisect $\mathbb{X}$. The only difficulty comes from the need to separate the ε-boxes from the others to ensure termination. So we need to remove ε-boxes from $\mathbb{X}$ and put them in a subpaving $\mathbb{E}$. Hence, the above iteration has to be replaced by:

$$(\mathbb{X} \cup \mathbb{E}) \leftarrow f(\mathbb{X} \cup \mathbb{E}).$$

We give now the algorithm. Figure 6 illustrates the process.

Algorithm 10: **invset** :: (SUBP,SUBP) $\rightarrow$ SUBP

```
1 invset (𝕏, 𝔼) = ;
2 if 𝕏 = ∅ then 𝔼 else
3     let  𝕐 = map f (𝕏∪𝔼);
4          𝕏₁ = 𝕐∩𝕏;
5          𝔼₁ = 𝕐∩𝔼;
6          (𝔼₂, 𝕏₂) = map** trace⟨P⟨ε⟩⟩ 𝕏₁;
7          𝕏₃ = map* bisect 𝕏₂  in invset (𝕏₃, (𝔼₁ ∪ 𝔼₂))
```

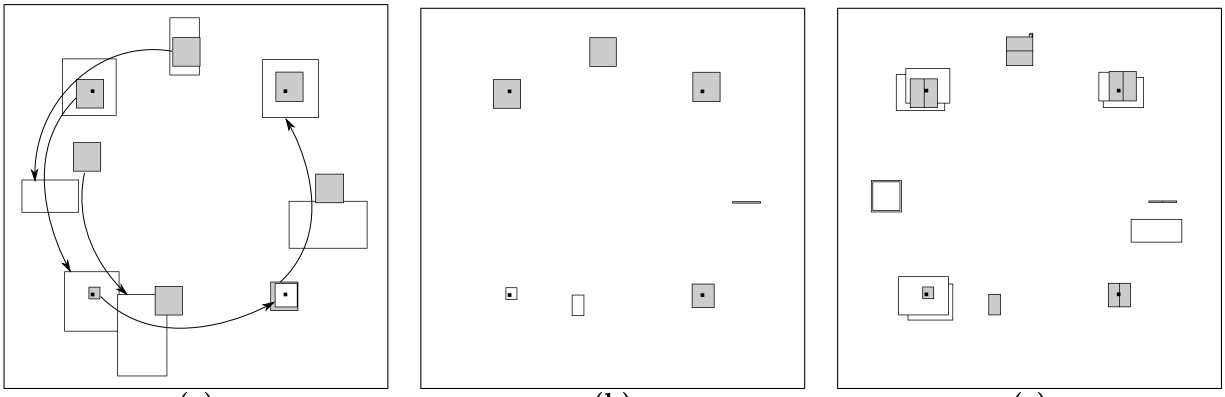


Figure 6: **The invariant set.** The function f is $(x_1, x_2) \mapsto \frac{1}{\|x\|}(x_2, -x_1)$. The initial set $\mathcal{S}^0$ is the 8 gray boxes shown in (a). The invariant set is made of 4 points, depicted in black on each picture. The pictures show the very first steps of the algorithm. **(a).** The white boxes represent the image of $\mathbb{X} = \mathcal{S}^0$ (arrow highlights the way a box is mapped to another). **(b).** The intersection of $\mathbb{X}$ and $f(\mathbb{X})$. The white boxes represent now the one that are put in $\mathbb{E}_2$. We can see that one box has already been discarded. **(c).** The gray subpaving is the union of $\mathbb{X}_3$ (i.e., after the bisection) and $\mathbb{E}_2$. The white boxes represent the image by f. It is clear that two other boxes will be discarded at the next iteration. The algorithm converges to a subpaving that encapsulates the four black points.

7 Quantified Constraints

A quantified constraint [14, 13, 4] is a formula in first-order logic, that is, an expression with occurrences of the quantifier symbols "$\exists$" and "$\forall$", variables bound to these symbols, domains associated to these variables and, at least, one free variable.

In this section, we show the benefits of programming with explicit pavings through the following case of study: find the set of $p \in [p]$ such that

$$\exists p_0 \in [p]_0,\ p_1 \in [p]_1 \ldots, p_{n-1} \in [p]_{n-1} \quad c_0(p_0, p_1) \wedge \ldots \wedge c_n(p_{n-1}, p). \tag{3}$$

where $[p]_0 \times \ldots \times [p]_{n-1} \times [p]$ is an input box and $c_0 \ldots, c_n$ regular constraints. This problem arises, for instance, in the mechanical design of a serial robot [7], as depicted in Figure 7. Note that this problem has a particular cascade structure. Indeed, if we introduce the constraint

$$c'_i(p_i) \iff \exists p_0 \in [p]_0, \ldots, p_{i-1} \in [p]_{i-1} \quad c_0(p_0, p_1) \wedge \ldots \wedge c_{i-1}(p_{i-1}, p_i)$$

then (3) becomes:

$$\exists p_i \in [p]_i \ldots, p_{n-1} \in [p]_{n-1} \quad c'_i(p_i) \wedge c_i(p_i, p_{i+1}) \wedge \ldots \wedge c_n(p_{n-1}, p). \tag{4}$$

This property plays a key role for our approach. To better understand why, let us first consider a naive strategy. We can consider the n variables $p_0, \ldots, p_{n-1}, p$ as a single vector of variables, pave the solution set of $c_0 \wedge \ldots \wedge c_n$ (a subset of $\mathbb{R}^{2n}$ in the case of 2-dimensional variables p_i like in Figure 7) and project it onto p. The cascade structure makes this approach inefficient. Indeed, the constraint c'_i as defined above is independent from $p_{i+1}, \ldots, p_{n-1}$ and p. This means that the corresponding solution set $\mathcal{S}_i$ can be paved first so that only the $(n-i+1)$-dimensional

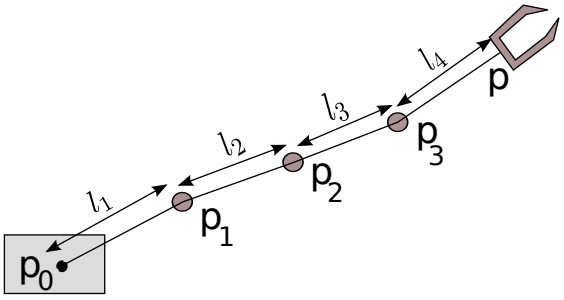


Figure 7: . **Workspace of a serial robot.** The position of the basepoint and the joints are the bound variables $p_0, \ldots, p_{n-1}$. The position of the effector p is the unknown. Each pair of points (p_i, p_{i+1}) is connected by a distance constraint $c_i(p_i, p_{i+1}) \iff \|p_i - p_{i+1}\| \leq l_{i+1}$. The basepoint of the robot is known to lie within a box $[p]_0$ (say, gotten from some measurement). The question is to determine the set of points that can be surely/potentially reached by the effector.

problem (4) is left to be solved. In the "top-down" approach, the reader should imagine that $\mathcal{S}_{n-1}$ is recalculated each time p is bisected and the phenomenon is repeated recursively.
The "bottom-up" approach is therefore much more adapted to the structure, provided, of course, that we are able to manage explicit pavings. As usual, we assume that for each constraint c_i inner and outer contractors C_i and $\overline{C}_i$ are available. Following the bottom-up approach, we also assume that a paving of $\mathcal{S}_i$ has already been calculated when p_{i+1} is to be paved. This includes an inner subpaving $\mathbb{K}_i$ and a boundary subpaving $\mathbb{E}_i$. Now, the only tricky part of the algorithm is to derive from C_i, $\overline{C}_i$, $\mathbb{K}_i$ and $\mathbb{E}_i$ an inner contractor $\overline{C}'_i$ and an outer contractor C'_i with respect to c'_i, i.e.,

$$\exists p_i \in \mathcal{S}_i \quad c_i(p_i, p_{i+1}).$$

A first way to define C'_i would be by using the *proj-union* operator introduced in [2] (see also [13] for a simpler variant). Figure 8.(a) illustrates how it works. However, this operator has serious drawbacks:

- It restricts the domain of the bound variable (here, p_i) to be a box $[p]_i$ whereas a set $\mathcal{S}_i$ (described by some paving) is required in our context.
- It runs itself a sub-paver. It is quite inelegant that a hard-coded paver is run silently in a framework where, precisely, pavers should be programmable.
- It is only based on the outer contractor of the embedded constraint c_i whereas, in our context, inner contractors are also available and should be incorporated to boost the sub-paver. See Figure 8.
- Only the bound variable p_i is bisected and with a very naive heuristic (the domain is split into a fixed number of slices).

First, we can use a set inversion to get efficient contractors C'_1 and $\overline{C}'_1$. For C'_1, we just have to project the paving obtained by applying C_0 and $\overline{C}_0$ to $[p]_0 \times b$ where $b \subseteq [p]_1$ is the input box (see Figure 8):

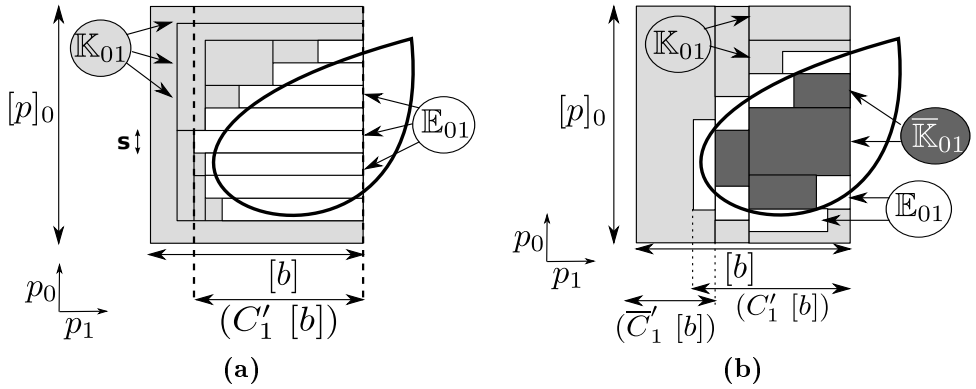


Figure 8: **Contractor for a quantified constraint.** Two possible implementations for C'_1. Variables p_0 and p_1 are in 1 dimension and $[b] \subseteq [p]_1$. **(a).** Using the *proj-union* operator. The product set $[p]_0 \times [b]$ is paved using the contractor C_0 for c_0. $[p]_0$ is split in slices (like **s**). The union of the ε-boxes of $\mathbb{E}_{01}$ are projected onto p_1. **(b).** Using the set inverse algorithm. We see the effect of an inner contractor inside the subpaver. Also, $[b]$ is contracted outwardly and inwardly.

$$\mathbf{C'_1}\ [\mathrm{b}] = \mathbf{let}\ (-, \overline{\mathbb{K}}_{01}, \mathbb{E}_{01}) = \mathbf{setinv}\langle C_0, \overline{C}_0, \varepsilon\rangle\ [p_0] \times [b]$$
$$\mathbf{in}\ \square\, \pi_{p_1}(\overline{\mathbb{K}}_{01} \cup \mathbb{E}_{01})$$

The definition of $\overline{C}'_1$ is very similar. The only difference is that only $\mathbb{K}_{01}$ is projected and that the set difference has to be returned instead, that is,

$$\square[b] \backslash \pi_{p_1}(\overline{\mathbb{K}}_{01}).$$

Now, $\overline{\mathbb{K}}_1$ and $\mathbb{E}_1$ can be obtained by using **setinverse** with C'_1 and $\overline{C}'_1$.

We can compute $\overline{\mathbb{K}}_{i+1}$ and $\mathbb{E}_{i+1}$ from $\overline{\mathbb{K}}_i$ and $\mathbb{E}_i$ exactly the same way. There is just a slight difference. Since the input of the subpaver is not a single box (as $[p]_0$) but a whole subpaving, we can use a simple **map** instead of **setinv**:

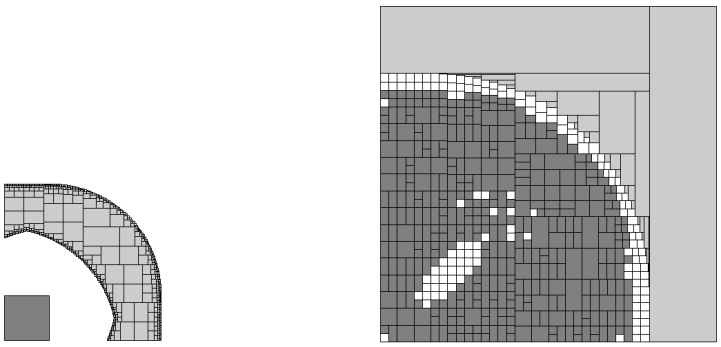
$$\begin{aligned} C'_{i+1}\ [b] &= \square\ (\mathbf{map}\ (\pi_{p_{i+1}} \circ C_i)\ (\mathbb{E}_i \cup \overline{\mathbb{K}}_i) \times [b]) \\ \overline{C}'_{i+1}\ [b] &= \square\ [b] \backslash (\mathbf{map}\ (\pi_{p_{i+1}} \circ \overline{C}_i)\ (\overline{\mathbb{K}_i} \times [b])) \end{aligned}$$

Figure 9 shows some result we have obtained.

8 Conclusion

In this paper, we have shown the predominance of using explicit search tree representations with continuous domains. We have also proposed a formalism to write algorithms based on such representation.

The main advantage is the ability to handle parts of the search space as Euclidean sets and to perform operations between these sets. Such operations are useful with continuous domains merely because solvers are asked to calculate subsets of the initial domain that fulfills a global property, rather than tuples that individually satisfy a property. Sometimes, these operations are even necessary to avoid an exponential blow-up of redundant computations, as we have shown in our last example on quantified constraints.



(a) **(b)**

Figure 9: **Result of the bottom-up approach** applied on the serial robot example with two arms. Each arm can only move in the positive orthant. **(a).** The box $[p]_0$ is in dark gray, on the lower left corner. The gray and white boxes depict the set $\mathcal{S}_1$. **(b).** The set $\mathcal{S}_2$ obtained from (a). The inner boxes are in dark gray. The light gray is the impossible area (the point p_2 cannot access).

This work is a first theoretical approach towards systems based on explicit pavings. A prototype has been developed to validate this approach but, of course, the current implementation has no pretension of efficiency whatsoever. Indeed, operations on subpavings, like the set difference, requires clever algorithms to be fast and compact (in the result provided). Such algorithms can be found in the realm of computational geometry [3] and have to be developed to get good performances. We have only implemented so far naive sweeping-like algorithm in our prototype since it was enough for the proof of concept. Note that all the figures of the paper has been generated by our prototype.

Our hope is that, in the future, our language and this prototype may serve as a basis for the development of a new generation of solvers.

References

[1] I. Braems, F. Berthier, L. Jaulin, M. Kieffer, and E. Walter. Guaranteed Estimation of Electrochemical Parameters by Set Inversion Using Interval Analysis. *Journal of Electroanalytical Chemistry*, 495(1):1–9, 2001.

[2] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173(11):1079–1100, 2009.

[3] M. De Berg, O. Cheong, and M. van Kreveld. *Computational geometry: algorithms and applications*. Springer, 2008.

[4] A. Goldsztejn and L. Jaulin. Inner and Outer Approximations of Existentially Quantified Equality Constraints. In *CP*, pages 198–212. Springer, 2006.

[5] A. Goldsztejn and L. Jaulin. Inner Approximation of the Range of Vector-Valued Functions. *Reliable Computing*, 14:1–23, 2010.

[6] E.R. Hansen. *Global Optimization using Interval Analysis*. Marcel Dekker, 1992.

[7] E.J. Haug, C.M. Luh, F.A. Adkins, and J.Y. Wang. Numerical Algorithms for Mapping Boundaries of Manipulator Workspaces. *ASME Journal of Mechanical Design*, 118:228–234, 1996.

[8] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

[9] L. Jaulin and E. Walter. Set Inversion via Interval Analysis for Nonlinear Bounded-Error Estimation. *Automatica*, 29(4):1053–1064, 1993.

[10] M. Lhommeau, L. Jaulin, and L. Hardouin. Capture Basin Approximation Using Interval Analysis. *International Journal of Adaptive Control and Signal*, 25(3):264–272, 2011.

[11] P.S.V. Nataraj, A.K. Prakash, and S. Srivastava. Improved Algorithm for Set Inversion With Application to a Jet Engine Control System. *Journal of Dynamic Systems, Measurement, and Control*, 127(1):163–166, 2005.

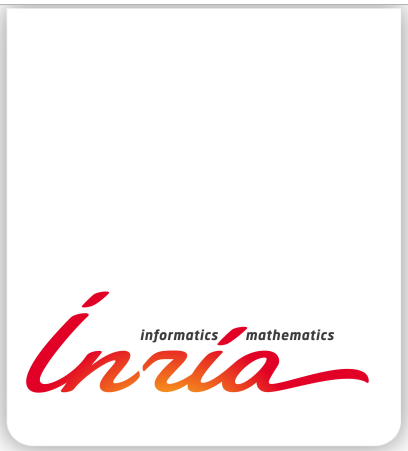
[12] A. Neumaier. *Complete Search in Continuous Global Optimization and Constraint Satisfaction*, pages 1–99. Cambridge Univ. Press, 2004.

[13] S. Ratschan. Efficient Solving of Quantified Inequality Constraints over the Real Numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.

[14] S.P. Shary. A New Technique in Systems Analysis Under Interval Uncertainty and Ambiguity. *Reliable Computing*, 8(5):321–418, 2002.

[15] G. Trombettoni, I. Araya, B. Neveu, and G. Chabert. Inner Regions and Interval Linearizations for Global Optimization. In *AAAI*, pages 99–104, 2011.

[16] P. Herrero Vinas, M. A. Sainz, J. Vehi, and L. Jaulin. Quantified set inversion algorithm with applications to control. *Reliable computing*, 11(5):369–382, 2006.



RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399