Learning Monotone Partitions of Partially-Ordered Domains

Nicolas Basset, Oded Maler*, José-Ignacio Requeno Jarabo
{bassetni, jose-ignacio.requeno-jarabo}@univ-grenoble-alpes.fr

VERIMAG, CNRS and Université Grenoble-Alpes, FRANCE

Abstract. We present an algorithm for learning the boundary between an upward-closed set $\overline{X}$ and its downward-closed complement. The algorithm selects sampling points for which it submits membership queries $x \in \overline{X}$ to an oracle. Based on the answers and relying on monotonicity, it constructs an approximation of the boundary. The algorithm generalizes binary search on the continuum from one-dimensional (and linearly-ordered) domains to multi-dimensional (and partially-ordered) ones. Applications include the approximation of Pareto fronts in multi-criteria optimization and parameter synthesis for predicates where the influence of parameters is monotone. The procedure explained in this paper has been implemented in a free and publicly available Python library and tested with several examples of varying dimension.

1 Introduction and Motivation

Let X be a bounded and partially ordered set that we consider from now on to be $[0,1]^n$. A subset $\overline{X}$ of X is *upward closed* in X if

$$\forall x, x' \in X \ (x \in \overline{X} \wedge x' \geq x) \rightarrow x' \in \overline{X}.$$

Naturally, the complement of $\overline{X}$, $\underline{X} = X - \overline{X}$ is downward closed, and we use the term *monotone bi-partition* (or simply partition) for the pair $M = (\underline{X}, \overline{X})$. We do not have an explicit representation of M and we want to approximate it based on queries to a membership oracle which can answer for every $x \in X$ whether $x \in \overline{X}$. Based on this information we construct an approximation of M by a pair of sets, $(\underline{Y}, \overline{Y})$ being, respectively, a downward-closed subset of $\underline{X}$ and an upward-closed subset of $\overline{X}$, see Figure 1. This approximation, conservative in both directions, says nothing about points residing in the gap between $\underline{Y}$ and $\overline{Y}$. This gap can be viewed as an over-approximation of $bd(M)$, the boundary between the two sets. There are two degenerate cases of monotone partitions, $(X, \emptyset)$ and $(\emptyset, X)$ that we ignore from now on, and thus assume that $\mathbf{0} \in \underline{X}$ and $\mathbf{1} \in \overline{X}$, where $\mathbf{r}$ denotes $(r, \ldots, r)$. We adopt the conventions that $bd(M)$ belongs to $\overline{X}$.

* Oded Maler passed away at the beginning of September 2018. This work that the two other of us finished was initiated by Oded Maler and mainly done in collaboration when he was still alive.

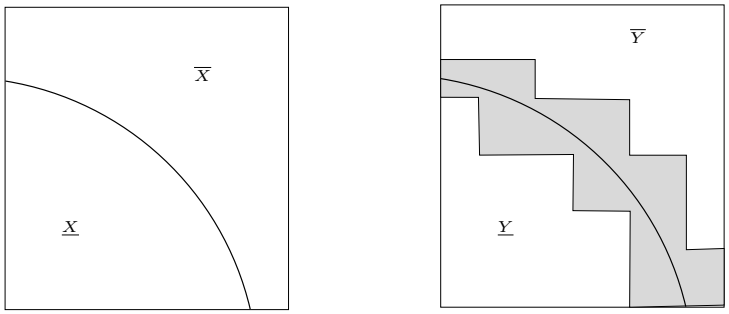


Fig. 1: A monotone partition and its approximation.

Before presenting the algorithmic solution that we offer to the problem, let us discuss some motivations. To start with, the problem is interesting for its own sake as a neat high-dimensional generalization of the problem of locating a boundary point that splits a straight line into two intervals. This problem is solved typically using binary (dichotomic) search, and indeed, the essence of our approach is in embedding binary search in higher dimension.

One major motivation comes from the domain of *multi-criteria optimization* where solutions are evaluated according to several criteria and the cost of a solution can be viewed as a point in a multi-dimensional cost space X. The optimal cost of such optimization problems is rarely a single point but rather a set of incomparable points also, known as the *Pareto front* of the problem. It consists of solutions that cannot be improved in one dimension without being worsened in another. Under certain assumptions, the Pareto front can be viewed as the boundary of a monotone partition. For a minimization problem, $\underline{X}$ corresponds to infeasible costs and $\overline{X}$ represents the feasible costs. The Pareto front is the set $bd(M) = \min(\overline{X})$ and the approximation that is provided is $\min(\overline{Y})$.

Another motivation comes from some classes of parametric identification problems. Consider a parameterized family of predicates/constraints $\{\varphi_p\}$ where p is a vector of parameters ranging over some parameter space. Given an element u from the domain of the predicates, we would like to know the range of parameters p such that $\varphi_p(u)$ holds. We say that a parameter p has a fixed (positive or negative) polarity if increasing its value will have a monotone effect on the set of elements that satisfy it. For example if a parameter p appears in a parameterized predicate $u \leq p$, then for any $p' > p$ and any u, $\varphi_p(u)$ implies $\varphi_{p'}(u)$. When no parameter appears in two constraints in opposing sides of an inequality, and after some pre-processing, the set of parameters that lead to satisfaction is upward closed. Its set of minimal elements indicates the set of tightest parameters that lead to satisfaction of $\varphi_p(u)$, which is a valuable information about u. In [1] we explored the idea for the domain of real-valued signals $u(t)$ and temporal formulas such as $\exists t < p_1 \ u(t) < p_2$.

1.1 Related work

In [4] we developed a procedure for computing such an approximation using a variant of binary search that submits queries to a constraint solver concerning the existence of solutions of a given cost x. The costs used in the queries were selected in order to reduce the distance between the boundaries of $\overline{Y}$ and $\underline{Y}$ and improve approximation quality. The present algorithm provides an alternative way to approximate Pareto fronts with the additive feature to provide a cloud of points of the Pareto front.

A preliminary version of the procedure that we propose in this paper (excluding materials of Section 4) has been implemented in [7] and successfully used for learning valuations of parametric STL specifications over time-series data [9,8].

2 Binary Search in One Dimension

Our major tool is the classical binary search over one-dimensional and totally-ordered domains, where a partition of $[0,1]$ is of the form $M = ([0,z),[z,1])$ for some $0 < z < 1$. The outcome of the search procedure is a pair of numbers $\underline{y}$ and $\overline{y}$ such that $\underline{y} < z < \overline{y}$, which implies a partition approximation $M' = ([0,\underline{y}),[\overline{y},1])$. The quality of M' is measured by the size of the gap $\overline{y} - \underline{y}$, which can be made as small as needed by running more steps. Note that in one dimension, $\overline{y} - \underline{y}$ is both the volume of $[\underline{y},\overline{y}]$ and its diameter. We are going to apply binary search to straight lines of arbitrary position and arbitrary positive orientation inside high-dimensional X, hence we formulate it in terms that will facilitate its application in this context.

Definition 1 (Line Segments in High-Dimension). *The line segment connecting two points $\underline{x} < \overline{x} \in X = [0,1]^n$ is their convex hull*

$$\langle \underline{x}, \overline{x} \rangle = \{(1-\lambda)\underline{x} + \lambda\overline{x} : \lambda \in [0,1]\}.$$

The segment inherits a total order from $[0,1]$: $x \leq x'$ whenever $\lambda \leq \lambda'$.

The input to the binary search procedure, written in Algorithm 1, is a line segment ℓ and an oracle for a monotone partition $M = (\underline{\ell}, \overline{\ell}) = (\langle \underline{x}, z\rangle, \langle z, \overline{x}\rangle)$, $\underline{x} < z < \overline{x}$. The output is a sub-segment $\langle \underline{y}, \overline{y}\rangle$ containing the boundary point z. The procedure is parameterized by an error bound $\epsilon \geq 0$, with $\epsilon = 0$ representing an ideal variant of the algorithm that runs indefinitely and finds the exact boundary point. Although realizable only in the limit, it is sometimes convenient to speak in terms of this variant. Fig. 2 illustrates several steps of the algorithm.

3 Monotone Partitions in Dimension 2

The following definitions are commonly used in multi-criteria optimization and in partially-ordered sets in general. Given two points $x = (x_1, \ldots, x_n)$ and $x' = (x'_1, \ldots, x'_n)$, we use the following notation $x \leq x'$ if $x_i \leq x'_i$ for every i; $x < x'$ if

Algorithm 1 One dimensional binary search: $search(\langle \underline{x}, \overline{x} \rangle, \epsilon)$

Input: A line segment $\ell = \langle \underline{x}, \overline{x} \rangle$, a monotone partition $M = (\underline{\ell}, \overline{\ell})$ accessible via an oracle $member()$ for membership in $\overline{\ell}$ and an error bound $\epsilon \geq 0$.
Output: A line segment $\langle \overline{y}, \underline{y} \rangle$ containing $bd(M)$ such that $\overline{y} - \underline{y} \leq \epsilon$.
$\langle \underline{y}, \overline{y} \rangle = \langle \underline{x}, \overline{x} \rangle$
while $\overline{y} - \underline{y} \geq \epsilon$ **do**
    $y = (\underline{y} + \overline{y})/2$
    if $member(y)$ **then**
        $\langle \underline{y}, \overline{y} \rangle = \langle \underline{y}, y \rangle$ ▷ left sub-interval
    else
        $\langle \underline{y}, \overline{y} \rangle = \langle y, \overline{y} \rangle$ ▷ right sub-interval
    end if
end while
return $\{\langle \underline{y}, \overline{y} \rangle\}$

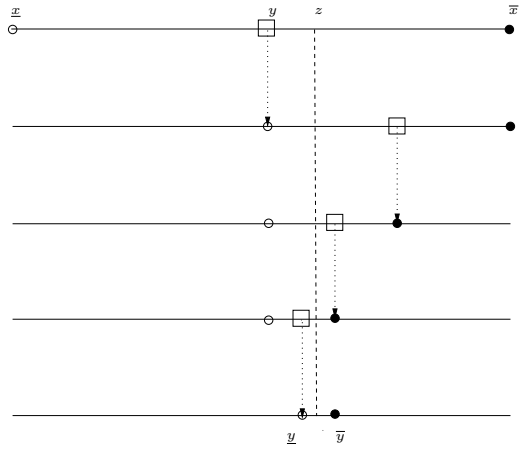


Fig. 2: Binary search and the successive reduction of the uncertainty interval.

$x \leq x'$ and $x_i < x'_i$ for some i and in this case we say that x dominates x'; $x||x'$ if $x \not\leq x'$ and $x' \not\leq x$, which means that $x <_i x'$ and $x' <_j x$ for some i and j. In this case we say that x and x' are incomparable.

Any two points $\underline{x} < \overline{x}$ define a *box* $\lfloor \underline{x}, \overline{x} \rceil = \{x : \underline{x} \leq x \leq \overline{x}\}$ for which they are, respectively, the minimal and maximal corners, as well as the endpoints of the diagonal $\langle \underline{x}, \overline{x} \rangle$. Equivalently the box $\lfloor \underline{x}, \overline{x} \rceil$ is defined as the product of the intervals $\prod_{i=1}^{n} [\underline{x}_i, \overline{x}_i]$. Its volume is $\texttt{Vol}(\lfloor \underline{x}, \overline{x} \rceil) = \prod_{i=1}^{n} (\overline{x}_i - \underline{x}_i)$. Two boxes can intersect (we say also overlap), the intersection being itself a box. We do not take care of the overlap of volume 0 and it will be often the case (even in the ideal version of our algorithms) that the points in the frontier of a box can be found in the frontier of several other boxes we consider.

The procedure for learning/approximating a monotone partition is written down in Algorithm 2 and works as follows. It maintains at any moment the cur-

rent approximation $(\underline{Y}, \overline{Y})$ of the partition as well as a list L of boxes whose union constitutes an over-approximation of the boundary, that is $bd(M) \in \cup\{\lfloor \underline{x}, \overline{x} \rceil : \lfloor \underline{x}, \overline{x} \rceil \in L\}$. For efficiency reasons, L is maintained in a decreasing order of volume. We successively take the largest box $\lfloor \underline{x}, \overline{x} \rceil$ from L, and find in it parts that can we move to $\underline{Y}$ and $\overline{Y}$. As the algorithm proceeds $\underline{Y}$ and $\overline{Y}$ augments and get closer to $\underline{X}$ and $\overline{X}$; and the total volume of the boundary approximation decreases until some stopping criterion is met (e.g. when going below a threshold). Some steps of the algorithm are illustrated in Fig. 3.

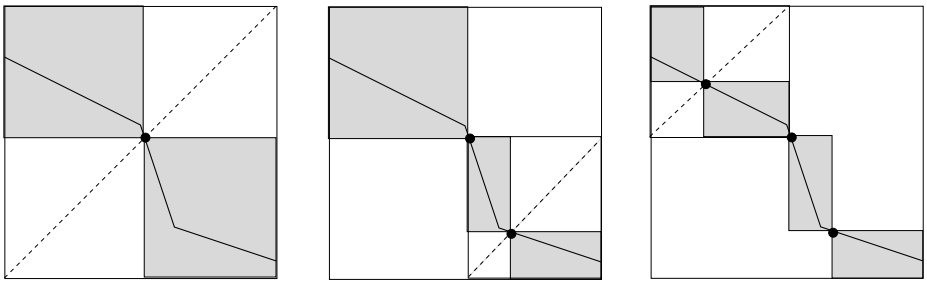


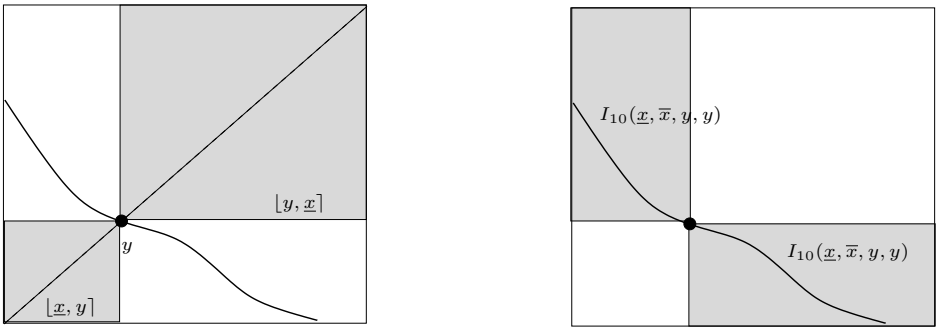
Fig. 3: Successive approximation of the partition boundary by running binary search on diagonals of incomparable boxes.

Algorithm 2 Approximating a monotone partition (and its boundary) by unions of rectangular cones.

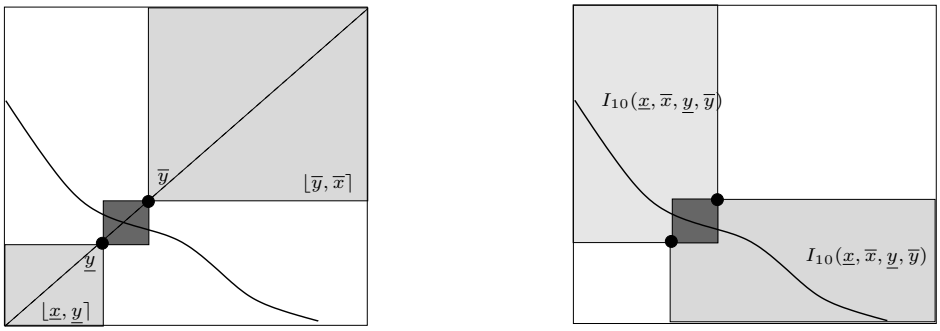
Input: A box $X = \lfloor \mathbf{0}, \mathbf{1} \rceil$, a partition $M = (\underline{X}, \overline{X})$ accessed by a membership oracle for $\overline{X}$ and an error bound δ.
Output: An approximation $M' = (\underline{Y}, \overline{Y})$ of M and an approximation L of the boundary $bd(M)$ such that $|L| \leq \delta$. All sets are represented by unions of boxes.
$L = \{X\}$; $(\underline{Y}, \overline{Y}) = (\emptyset, \emptyset)$ ▷ initialization
repeat
  pop first $\lfloor \underline{x}, \overline{x} \rceil \in L$ ▷ take the largest box from the boundary approximation
  $\langle \underline{y}, \overline{y} \rangle = search(\langle \underline{x}, \overline{x} \rangle, \epsilon)$ ▷ run binary search on the diagonal
  $\overline{Y} = \overline{Y} \cup \{\lfloor \overline{x}, \overline{y} \rceil\}$ ▷ add dominated sub-box
  $\underline{Y} = \underline{Y} \cup \{\lfloor \underline{x}, \underline{y} \rceil\}$ ▷ add dominating sub-box
  $L = L \cup I(\overline{x}, \underline{x}, \overline{y}, \underline{y})$ ▷ insert remainder of $\lfloor \underline{x}, \overline{x} \rceil$ to L
  $\texttt{Vol}(L) = \texttt{Vol}(X) - \texttt{Vol}(\underline{Y}) - \texttt{Vol}(\overline{Y})$
until $\texttt{Vol}(L) \leq \delta$

To treat the box $\lfloor \underline{x}, \overline{x} \rceil$ we rely on the observation that any line ℓ of a positive slope inside a box $\lfloor \underline{x}, \overline{x} \rceil$ that admits a monotone partition M, intersects $bd(M)$ at most once. In particular, the diagonal $\ell = \langle \underline{x}, \overline{x} \rangle$ of the box is guaranteed to intersect $bd(M)$ and such intersection is over-approximated by the segment

$\langle \underline{y}, \overline{y} \rangle$ returned by the one-dimensional binary search algorithm applied on the diagonal $\langle \underline{x}, \overline{x} \rangle$. In this case $\lfloor \underline{x}, \underline{y} \rceil$ is added to $\underline{Y}$ since its points dominate $\overline{y}$ and the box $\lfloor \overline{y}, \overline{x} \rceil$ is added to $\overline{Y}$ since its points are dominated by $\underline{y}$. The remaining part of $\lfloor \underline{x}, \overline{x} \rceil$ stays in the border approximation and is split into a union of boxes $I(\overline{x}, \underline{x}, \overline{y}, \underline{y})$ described below. Fig. 4 illustrates the interaction between the result of the one-dimensional search process on the diagonal of a box $\lfloor \underline{x}, \overline{x} \rceil$ and the approximation of the higher-dimensional partition.



(a)



(b)

Fig. 4: (a) The effect of finding the exact intersection of the diagonal with the boundary; (b) The effect of finding an interval approximation of that intersection.

Definition 2 (Sub-intervals). *The sub-interval of the interval* $[\underline{x}, \overline{x}]$ *induced by* $a \in \{0, 1\}$ *and the interval* $[\underline{y}, \overline{y}] \subseteq [\underline{x}, \overline{x}]$ *is*

$$I_a(\underline{x}, \overline{x}, \underline{y}, \overline{y}) = [\underline{x}, \overline{y}] \text{ if } a = 0 \text{ and } [\underline{y}, \overline{x}] \text{ if } a = 1.$$

In the previous definition a is a boolean variable that evaluates to 1 (resp. 0) when the upper (resp. lower) part of the interval is selected. The previous definition extends to boxes (that is product of intervals) when a binary word $\alpha = \alpha_1 \cdots \alpha_n$ is provided:

Definition 3 (Sub-boxes). *Let $\alpha \in \{0,1\}^n$ and and 4 n-dimensional points $\underline{x} \leq \underline{y} \leq \overline{y} \leq \overline{x}$. The sub-boxes of $\lfloor \underline{x}, \overline{x} \rceil$ induced by $\lfloor \underline{y}, \overline{y} \rceil$ and α is*

$$B_\alpha(\underline{x}, \overline{x}, \underline{y}, \overline{y}) = \prod_{i=1}^{n} I_{\alpha_i}(\underline{x}_i, \overline{x}_i, \underline{y}_i, \overline{y}_i)$$

We define $I(\overline{x}, \underline{x}, \overline{y}, \underline{y})$ as the set of $B_\alpha(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ for $\alpha \in \{0,1\}^n \setminus \{0^n, 1^n\}$. Observe that the union of boxes in $I(\overline{x}, \underline{x}, \overline{y}, \underline{y})$ is equal to $\lfloor \underline{x}, \overline{x} \rceil \setminus \lfloor \underline{x}, \underline{y} \rceil \cup \lfloor \overline{y}, \overline{x} \rceil$, that is the part that remains in the boundary approximation when we discover $\lfloor \underline{y}, \overline{y} \rceil$ around a point of the boundary in $\lfloor \underline{x}, \overline{x} \rceil$. Note that these boxes overlap but the volume of the overlap is proportional to ϵ (because it is made of boxes each one having a side with length smaller than ϵ). Hence this overlap can be made arbitrarily small by decreasing the parameter ϵ. This is efficient as the number of queries in a one-dimensional binary search is logarithmic in $1/\epsilon$.

4 Monotone Partitions in Dimension Higher than 2

The goal of introducing changes to the learning procedure in higher dimensions is twofold. First we decompose into fewer boxes the part that stay in the boundary approximation (Sect. 4.1). Second we avoid a pitfall we identify in Sect. 4.2 by propagating the dominance to the boxes in the boundary approximation every time a point of the Pareto front is discovered. We sum up the new version of the algorithm in Sect. 4.3.

4.1 Enhanced partitioning step

In this section we come back to the treatment of the remainder of the box $\lfloor \underline{x}, \overline{x} \rceil$ when the two boxes above $\underline{y}$ and below $\overline{y}$ are removed. This remainder part was cover by the union of boxes of $I(\overline{x}, \underline{x}, \overline{y}, \underline{y})$. In two dimensions, there are only two boxes in this set and every point of the box $B_{01}(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ is incomparable to all the points of $B_{10}(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ except for the small overlapping part.

In higher dimension some boxes can be grouped together into bigger boxes. In particular when two sub-boxes differs along only one dimension, they are *adjacent*: their union is again a box. We introduce an extra the symbol $\star$ to encode that we do not care of this dimension. Defs 2–3 must be consequently extended for supporting the symbol $\star$ so that $I_\star(\underline{x}, \overline{x}, \underline{y}, \overline{y}) = [\underline{x}, \overline{x}]$ in one dimension, and $B_\alpha(\underline{x}, \overline{x}, \underline{y}, \overline{y}) = \prod_{i=1}^{n} I_{\alpha_i}(\underline{x}_i, \overline{x}_i, \underline{y}_i, \overline{y}_i)$ with $\alpha \in \{0,1,\star\}^n$ in higher dimension. Examples of boxed created by these expressions are given in Fig. 5.

A partition of the index set $A = \{0,1\}^n \setminus \{0^n, 1^n\}$ is a set E of non-empty subsets B such that $A = \cup_{B \in E} B$ and $B \cap B' = \emptyset$ for every two different parts

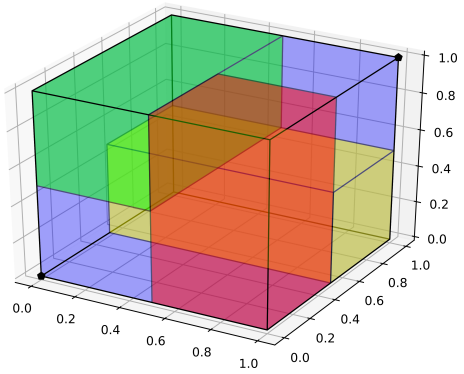


Fig. 5: A way of decomposing the remainder part into $2n-3=3$ boxes instead of $2^n-2=6$ in dimension $n=3$. Each coloured box corresponds to a box $B_{\alpha_{\texttt{color}}}(\underline{x},\overline{x},\underline{y},\overline{y})$ with $\alpha_{\texttt{red}}=10\star$, $\alpha_{\texttt{yellow}}=\star 10$ and $\alpha_{\texttt{green}}=0\star 1$.

$B, B' \in E$. A partition E of the index set A is called *feasible* if for each part $B \in E$, the union of boxes of B is itself a box. Such a box can be described by a word $\beta \in \{0,1,\star\}^n$, that is $B_\beta(\underline{x},\overline{x},\underline{y},\overline{y}) = \cup_{\alpha\in B} B_\alpha(\underline{x},\overline{x},\underline{y},\overline{y})$. It will be convenient to confuse the notation and write β for the part B. For instance $E_3 = \{10\star, \star 10, 0\star 1\}$ is a feasible partition of $\{0,1\}^3 \setminus \{0^3, 1^3\}$ that encodes the boxes represented in Fig. 5. In the following proposition we use the notation $\star E_n = \{\star\alpha' : \alpha' \in E_n\}$ defining for instance

$$E_4 = \star\{10\star, \star 10, 0\star 1\} \cup \{1000\} \cup \{0111\} = \{\star 10\star, \star\star 10, \star 0\star 1, 1000, 0111\}.$$

Proposition 1. *Let $n \geq 3$. There exists a feasible partition E_n of $\{0,1\}^n \setminus \{0^n, 1^n\}$ with $2n-3$ parts defined recursively as follows:*

$$E_3 = \{10\star, \star 10, 0\star 1\} \text{ and for } n \geq 3, E_{n+1} = \star E_n \cup \{10^n\} \cup \{01^n\}.$$

There is no feasible partition with fewer number of elements.

Finally, in dimension $n > 2$, when updating the boundary approximation, we use the set $I'(\overline{x},\underline{x},\overline{y},\underline{y}) = \cup_{\alpha\in E_n}\{B_\alpha(\underline{x},\overline{x},\underline{y},\overline{y})\}$ containing $2n-3$ boxes instead of the set $I(\overline{x},\underline{x},\overline{y},\underline{y})$ containing 2^n-2 boxes.

Example 1. Consider again the Fig. 5. We run the idealised version of our algorithm and after the first call to the binary search over the diagonal discovers that the point $(0.5, 0.5, 0.5)$ is in the boundary. We remove the dominated part $\lfloor(0,0,0),(0.5,0.5,0.5)\rceil$ and the dominating part $\lfloor(0.5,0.5,0,5),(1,1,1)\rceil$, we then partition the remaining part into the three boxes show in Fig. 5. The three boxes having the same volume, we choose arbitrarily to treat the red box first.

4.2 Propagating dominance to boxes of the boundary approximation

In dimension higher than 2, it can be the case that when an approximated point $(\underline{y}, \overline{y})$ is found within a box $\lfloor \underline{x}, \overline{x} \rceil$, the downward closure of $\underline{y}$ intersects other boxes of the boundary approximation (see Example 2). These parts must also be removed from the boundary approximation to keep $\overline{Y}$ downward-closed. A symmetric remark holds for the upward-closure of $\overline{y}$.

Example 2 (Example 1 continued). When calling the binary search to the red box, we discover that the point $y' = (0.9, 0.2, 0.9)$ is in the boundary. We remove $[0.5, 0.9] \times [0, 0.2] \times [0, 0.9)$ and $[0.9, 1] \times [0.2, 0.5] \times [0.9, 1]$ from the red box and partition the remainder of the red box. Though there is a part outside the red box that also should be removed that is the intersection of the green box with the downward closure of y', that is $[0, 0.5] \times [0, 0.2] \times [0.5, 0.9]$.

As we saw in the example above we need to intersect downward closure of point with other boxes. For this the following proposition will be useful.

Proposition 2. *The downward closure of a point $\underline{y}$ intersect a box $\lfloor \underline{z}, \overline{z} \rceil$ iff $\min(\underline{y}, \overline{z}) \geq \underline{z}$ where the minimum is taken component-wise that is $[\min(\underline{y}, \overline{z})]_i = \min(\underline{y}_i, \overline{z}_i)$ for every i. When this condition is met then*

$$\lfloor \mathbf{0}, \underline{y} \rceil \cap \lfloor \underline{z}, \overline{z} \rceil = \lfloor \underline{z}, \min(\underline{y}, \overline{z}) \rceil = B_\beta(\underline{z}, \overline{z}, \underline{y}, \underline{y})$$

with $\beta_i = 0$ if $\underline{y}_i < \overline{z}_i$ and $\beta_i = \star$ otherwise.

When discovering a couple of point $(\underline{y}, \overline{y})$ we check for every box $\lfloor \underline{z}, \overline{z} \rceil$ of the boundary approximation if it intersects $\lfloor \mathbf{0}, \underline{y} \rceil$. If this is the case we add $\lfloor \mathbf{0}, \underline{y} \rceil \cap \lfloor \underline{z}, \overline{z} \rceil$ to $\overline{Y}$ and replace in the boundary approximation $\lfloor \underline{z}, \overline{z} \rceil$ by a set of boxes $I_{dwc}(\underline{z}, \overline{z}, \underline{y})$ whose union equals the difference $\lfloor \underline{z}, \overline{z} \rceil \setminus \lfloor \mathbf{0}, \underline{y} \rceil$.

Now we explain how to define $I_{dwc}(\underline{z}, \overline{z}, \underline{y})$ with the smallest number of boxes. This boxes must have an empty intersection with $B_\beta(\underline{z}, \overline{z}, \underline{y}, \underline{y})$. First of all, we need to choose only the coordinates for which $\overline{y}_i < \underline{z}_i$, the other being $\star$. Second, the other coordinates must not all be equal to 0. So it suffices to fill the coordinates for which $\overline{y}_i < \underline{z}_i$, denote m their cardinality, by the word of the set $\{0^k 1 \star^{m-k-1} : k = 0..m-1\}$. This gives m boxes and one can show that this number is minimal.

Example 3. Consider the point $\underline{y} = (0.2, 0.6, 0.2, 0.2)$ and the box $\lfloor \underline{z}, \overline{z} \rceil$ defined by $\underline{z} = (0.0, 0.0, 0.0, 0.0)$ and $\overline{z} = (0.5, 0.5, 0.5, 0.5)$ The second coordinate is the only one for which $\overline{y}_i \geq \underline{z}_i$ and hence $m = 3$. Then

$$\lfloor \mathbf{0}, \underline{y} \rceil \cap \lfloor \underline{z}, \overline{z} \rceil = B_{0\star00}(\underline{x}, \overline{z}, \underline{y}, \underline{y}) = [0.0, 0.2] \times [0.0, 0.5] \times [0.0, 0.2] \times [0.0, 0.2]$$

and $I_{dwc}(\underline{z}, \overline{z}, \underline{y}) = \cup_{\alpha \in \{1\star\star\star, 0\star1\star, 0\star01\}} \{B_\alpha(\underline{z}, \overline{z}, \underline{y}, \underline{y})\}$.

$$B_{1\star\star\star}(\underline{z}, \overline{z}, \underline{y}, \underline{y}) = [0.2, 0.5] \times [0.0, 0.5] \times [0.0, 0.5] \times [0.0, 0.5]$$

$$B_{0\star1\star}(\underline{z}, \overline{z}, \underline{y}, \underline{y}) = [0.0, 0.2] \times [0.0, 0.5] \times [0.2, 0.5] \times [0.0, 0.5]$$

$$B_{0\star01}(\underline{z}, \overline{z}, \underline{y}, \underline{y}) = [0.0, 0.2] \times [0.0, 0.5] \times [0.0, 0.2] \times [0.2, 0.5]$$

The same kinds of reasoning can be done for the upward closure of $\overline{y}$ and $I_{upc}(\underline{z}, \overline{z}, \overline{y})$ can be defined in a similar manner of $I_{dwc}(\underline{z}, \overline{z}, \underline{y})$.

4.3 Combining ideas in an updated algorithm

Algorithm 3 gathers all the considerations presented in this section. It shows how to mitigate the block explosion in the boundary because of dimensionality and fragmentation. To this end, it concatenates adjacent incomparable boxes (lines 12,16,19). It also shows how the propagation of dominance avoids unnecessary calls to the oracle, which could potentially be the slower part of the program.

In a generic situation, the precision of the binary search has an error $\epsilon > 0$ and, consequently, some incomparable boxes in the boundary may overlap. The overlapping areas must be taken into account when moving boxes directly from the frontier to a closure $\underline{Y}$,$\overline{Y}$, while this peculiarity did not happen in Algorithm 2 and need not to be explicitly considered. For the sake of readability and concision, Algorithm 3 omits the code that checks and prevents the insertion of redundant portions of boxes caused by overlapping. In any case, the accumulated deviation resulting from this with respect to the volume is negligible as long as ϵ is very small.

Algorithm 3 Approximating a monotone partition (and its boundary) by unions of rectangular boxes.

1: **Input**: A box $X = \lfloor \mathbf{0}, \mathbf{1} \rceil$, a partition $M = (\underline{X}, \overline{X})$ accessed by a membership oracle for $\overline{X}$ and an error bound δ.
2: **Output**: An approximation $M' = (\underline{Y}, \overline{Y})$ of M and an approximation L of the boundary $bd(M)$ such that $|L| \leq \delta$. All sets are represented by unions of boxes.
3: $L = \{X\}$; $(\underline{Y}, \overline{Y}) = (\emptyset, \emptyset)$ ▷ initialization
4: **repeat**
5: **pop** first $\lfloor \underline{x}, \overline{x} \rceil \in L$ ▷ take the largest box from the boundary approximation
6: $\langle \underline{y}, \overline{y} \rangle = search(\langle \underline{x}, \overline{x} \rangle, \epsilon)$ ▷ run binary search on the diagonal
7: $\underline{Y} = \underline{Y} \cup \{\lfloor \underline{x}, \underline{y} \rceil\}$ ▷ add dominating sub-box
8: $\overline{Y} = \overline{Y} \cup \{\lfloor \overline{x}, \overline{y} \rceil\}$ ▷ add dominated sub-box
9: **for** $\lfloor \underline{z}, \overline{z} \rceil \in L$ **do**
10: **if** $\lfloor \underline{z}, \overline{z} \rceil \cap \lfloor \overline{y}, \mathbf{1} \rceil \neq \emptyset$ **then** ▷ if intersects the upward-closure of $\overline{y}$
11: add $\lfloor \underline{z}, \overline{z} \rceil \cap \lfloor \overline{y}, \mathbf{1} \rceil$ to $\overline{Y}$; ▷ add dominated sub-box
12: replace $\lfloor \underline{z}, \overline{z} \rceil$ by $I_{upc}(\underline{z}, \overline{z}, \overline{y})$ in L; ▷ update the boundary
13: **end if**
14: **if** $\lfloor \underline{z}, \overline{z} \rceil \cap \lfloor \mathbf{0}, \underline{y} \rceil \neq \emptyset$ **then** ▷ if intersects the downward-closure of $\underline{y}$
15: add $\lfloor \underline{z}, \overline{z} \rceil \cap \lfloor \mathbf{0}, \underline{y} \rceil$ to $\underline{Y}$; ▷ add dominating sub-box
16: replace $\lfloor \underline{z}, \overline{z} \rceil$ by $I_{dwc}(\underline{z}, \overline{z}, \underline{y})$ in L; ▷ update the boundary
17: **end if**
18: **end for**
19: $L = L \cup I(\underline{x}, \overline{x}, \underline{y}, \overline{y})$ ▷ insert to L the sub-boxes incomparable to $\overline{y}$
20: $\texttt{Vol}(L) = \texttt{Vol}(X) - \texttt{Vol}(\underline{Y}) - \texttt{Vol}(\overline{Y})$
21: **until** $\texttt{Vol}(L) \leq \delta$

5 ParetoLib Library

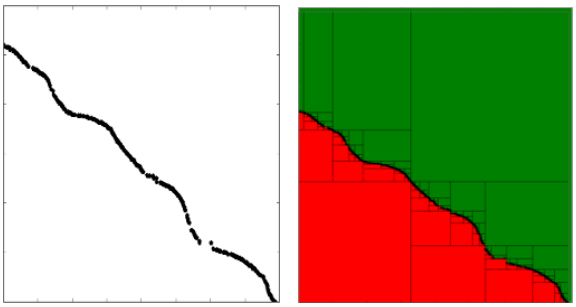
5.1 Overview

The procedure explained in this paper has been packaged in a Python library. The package is free and publicly available on Internet [2]. It has been implemented following the software engineering standards, suggestions and best practices for this language, including a brief documentation and a set of illustrative examples of how to use it. The library is compatible with Python 2.7, 3.4 or higher; and it is PEP 8 compliant. The code has been exhaustively tested, reaching a coverage of more than 90% of the code in the module devoted to the sequential approximation of the monotone partitions and boundary. It also supports multicore CPUs in order to take advantage of concurrent processing (i.e., the main algorithm processes nproc boxes from the boundary in parallel at each iteration step, being nproc the number of processing units). A preliminary version of our Pareto front discovery algorithm was implemented in [7] and demonstrated the usefulness in [8,9], but that tool did not include all the recent refinements and optimisations yet (section 4).

In our library, the oracle has been designed as an abstract interface that encapsulates the set of minimum operations that every prophet should provide. Later on, the oracle can be specialized for multiple application domains by simply implementing the abstract interface. For the moment, our library supports three specialized oracles for inferring the Pareto front.

- The OracleFunction, which defines the boundary between the upper and lower closures based on polynomial constraints. For instance in a 2D space, the border $x_1^2 + x_2^2 = 1$ contains all the points x = (x_1, x_2) in the surface of a sphere of radius one. Every point whose coordinates satisfy $x_1^2 + x_2^2 > 1$ falls in the upper closure, and, conversely, every point $x_1^2 + x_2^2 < 1$ in the lower closure. This oracle has been created as a 'proof of concept' for testing and debugging purposes.
- The OraclePoint, which defines the boundary between the upper and lower closures based on a discrete cloud of points. The cloud of points is saved in a NDTree [3], a data structure that is optimised for storing a Pareto front by removing redundant non-dominating points from the surface. A point x that dominates every member of the Pareto front belongs to the upper closure, while a point x that is dominated by any element of the Pareto front will fall in the lower closure.
- The OracleSTL, which defines the membership of a point x to the upper or lower closures based on the success in evaluating a Signal Temporal Logic (STL) [5] formula over a signal. The STL formula is parametrized with a set of variables which correspond to the coordinates of the point x (i.e., the number of parameters in the STL formula is equal to the dimension of x). Every point x satisfying the STL formula belongs to the upper closure, while every point x falsifying it will fall in the lower closure. Internally, OracleSTL asks queries to the JAMT tool [6].

Fig. 6 shows a set of discrete points that outlines the Pareto front, and the partitioning that is learnt by our algorithm thanks to the oracle guidance. The green side represents the upper closure and the red side represents the lower closure. A gap in blue may appear between the two closures, which is the border and can be set arbitrarily small depending on the accuracy required by the user.



(a) (b)

Fig. 6: Original Pareto front (a) and the inferred closures (b) learnt by our partitioning algorithm

5.2 Experiments

In this section we present the feasibility of our approach and the capabilities of our tool. To this end, we apply our procedure for discovering the Pareto front in two families of examples, whose boundary we already know a priori. These examples consist on the identification of a border outlined by the surface of a simplex; and the identification of a border outlined by the surface of a hypersphere. We consider simplexs and hyperspheres hosted in spaces of range $[0, 1]^d$ and dimension d from 2 to 5.

The surface that divides the space in two closures is implicitly defined by a cloud of points that are homogeneously sampled from a random point generator. Then, a OraclePoint will guide the process through all this section. Fig. 7–8 show the set of 1000 points that are used for specifying the simplex and the hypersphere in 3D, as well as the border that is discovered by our algorithm after running 200 steps. Red points are corners of the lower closure, and green points are corners of the upper closure. The maximal width of the boundary (i.e., distance between a green and a red point) is set to $\epsilon = 1e^{-5}$ for 3D.

Alternatively, Fig. 9 shows the convergence speed of the volume contained in each closure in 2D with respect to the steps of our algorithm. Increasing the dimension of the space has a direct impact on the convergence, as it can be seen in Fig. 10.

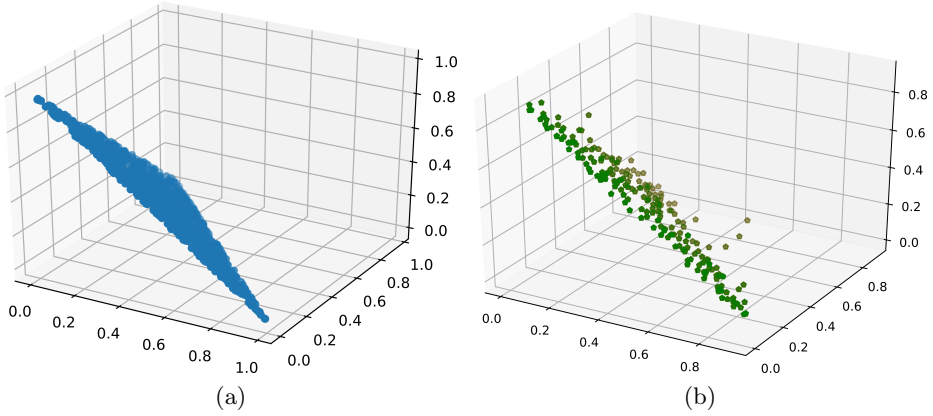


Fig. 7: Set of random points over the surface of a 3D simplex (a) and the inferred Pareto front (b)

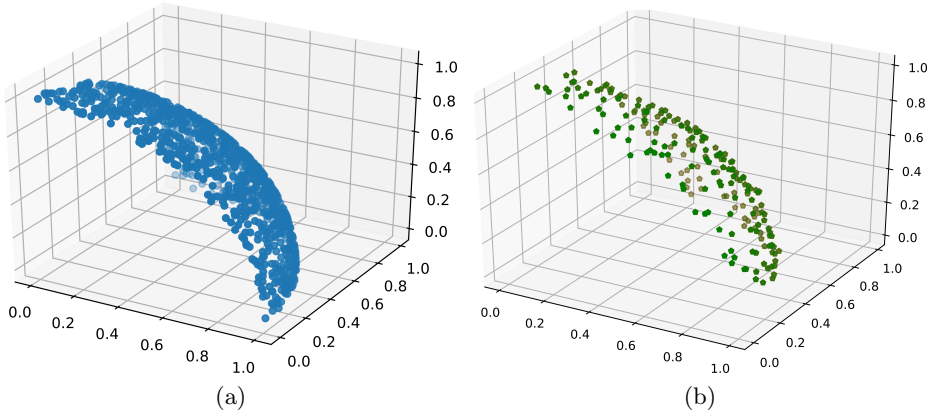


Fig. 8: Set of random points over the surface of a 3D hypersphere (a) and the inferred Pareto front (b)

Propagating the dominance of each new discovered Pareto point to the rest of remaining boxes in the boundary partially mitigates the dimensionality problem. However, it introduces a penalty in the computational speed of the tool because it inherently involves more operations per step. Table 1 shows the execution time (in seconds) consumed for running 1000 steps. The main source of this penalty comes from the fragmentation of the boundary in a huge number of very small cubes, which must be periodically polled after the discovery of every new Pareto point. The propagation of the dominance is only interesting when the cost of asking a membership query to the oracle consumes more time than inspecting the whole list of boxes in the border, and, therefore, reducing the number of

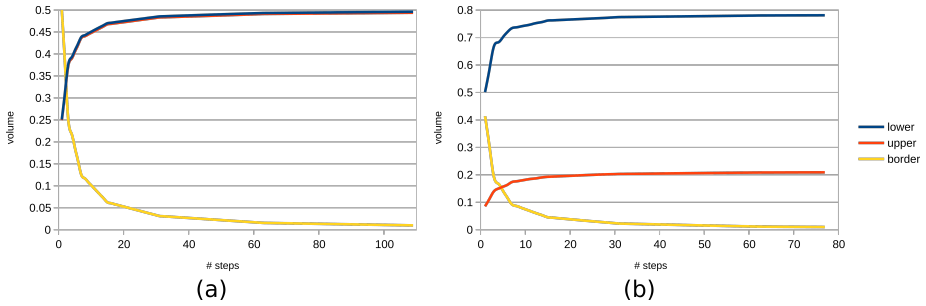


Fig. 9: Volume contained in the boundary, upper and lower closures for a 2D simplex (a) and hypersphere (b)

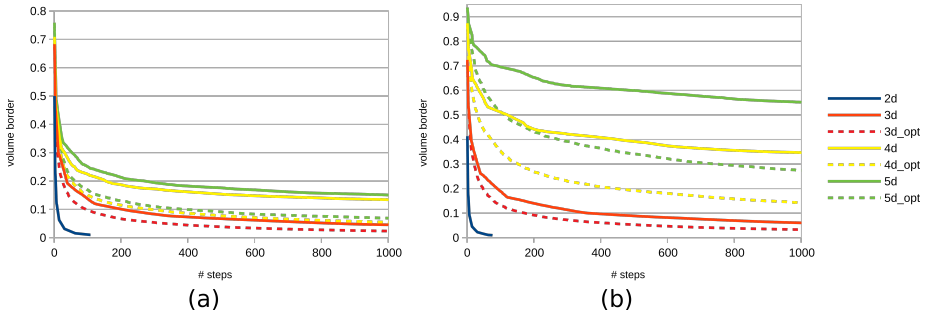


Fig. 10: Volume contained in the boundary of a simplex (a) and hypersphere (b) using the standard and optimised algorithms (opt. algorithm is not applicable to 2D)

membership queries becomes imperative. For instance, this situation happens for the OracleSTL, which must solve a STL formula per query.

	2d	3d	3d_opt	4d	4d_opt	5d
hypersphere	4.12	66.51	124.57	71.54	1017.65	74.26
simplex	5.69	62.55	123.59	70.88	1130.68	73.71

Table 1: Execution time in seconds consumed by the ParetoLib tool

In any case, the performance of our library is still good enough for small and medium size problems. It promises a good accuracy result within a reasonable amount of time and using affordable hardware resources. We must emphasize that almost half of the time spent in the simulation for the standard version of our algorithm is consumed by the oracle for solving the membership queries, implying that it is the main technical limitation factor together with the convergence problem associated to dimensionality. All the experiments are run in a single

core of a Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz with 8GBytes RAM, Linux Debian 8 and Python 2.7.

6 Conclusions

In this paper, we have presented an algorithm for learning the boundary (i.e., Pareto front) between an upward-closed set $\overline{X}$ and its downward-closed complement. The algorithm is based on an external oracle that answers membership queries $x \in \overline{X}$. According to the answers and relying on monotonicity, it constructs an approximation of the boundary. The algorithm generalizes binary search on the continuum from one-dimensional (and linearly-ordered) domains to multi-dimensional (and partially-ordered) ones. To this end, our algorithm divides the multi-dimensional space into multiple smaller blocks that are exhaustively explored until a stopping criterion is met (e.g. the volume of the uncertain part is smaller than a threshold).

Besides, the procedure explained in this paper has been implemented in a free and publicly available Python library. It has been tested with several examples of varying dimension; for instances, checking the membership of points to an hypersphere or a simplex from dimension 2 to 4.

We plan as future work, to apply our tool to various problem including parametric identification for STL formula and multicriteria optimisation. We would like to further ameliorate the tool and present its characteristics and performances in a dedicated tool paper.

References

1. Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *RV*, pages 147–160, 2011.
2. Nicolas Basset, Oded Maler, and José-Ignacio Requeno Jarabo. ParetoLib library. `https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/multidimensional_search`, 2018.
3. Andrzej Jaszkiewicz and Thibaut Lust. Nd-tree-based update: a fast algorithm for the dynamic non-dominance problem. *IEEE Transactions on Evolutionary Computation*, 2018.
4. Julien Legriel, Colas Le Guernic, Scott Cotton, and Oded Maler. Approximating the Pareto front of multi-criteria optimization problems. In *TACAS*, pages 69–83, 2010.
5. Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.
6. Dejan Ničković, Olivier Lebeltel, Oded Maler, Thomas Ferrère, and Dogan Ulus. Amt 2.0: Qualitative and quantitative trace analysis with extended signal temporal logic. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 303–319. Springer, 2018.
7. Marcell Vazquez-Chanlatte. Multidimensional thresholds. `https://github.com/mvcisback/multidim-threshold`, 2018.
8. Marcell Vazquez-Chanlatte, Jyotirmoy V. Deshmukh, Xiaoqing Jin, and Sanjit A. Seshia. Logical clustering and learning for time-series data. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 305–325. Springer, 2017.
9. Marcell Vazquez-Chanlatte, Shromona Ghosh, Jyotirmoy V. Deshmukh, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Time-series learning using monotonic logical properties. In Christian Colombo and Martin Leucker, editors, *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, volume 11237 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2018.