Viabilitree: A kd-tree Framework for Viability-based Decision

Isabelle Alvarez, Romain Reuillon, Ricardo de Aldama

Viabilitree: A kd-tree Framework for Viability-based Decision

Isabelle Alvarez[a,b,*], Romain Reuillon[c], Ricardo de Aldama[c]

[a] *Irstea, UR LISC Laboratoire d'ingénierie des systèmes complexes, Aubière, France*
[b] *UPMC Univ Paris 06, LIP6, Paris, France*
[c] *ISC-PIF, Paris, France*

Abstract

The mathematical viability theory offers concepts and methods that are suitable to study the compatibility between a dynamical system described by a set of differential equations and constraints in the state space. The result sets built during the viability analysis can give very useful information regarding management issues in fields where it is easier to discuss constraints than objective functions. However, computational problems arise very quickly with the number of state variables, and the practical implementation of the method is difficult, although there exists a convergent numerical scheme and several approaches to bypass the computational problems. In order to popularize the use of viability analysis we propose a framework in which the viability sets are represented and approximated with particular kd-trees. The computation of the viability kernel is seen as an active learning problem. We prove the convergence of the algorithm and assess the approximation it produces for known problems with analytical solution. This framework aims at simplifying the declaration of the viability problem and provides useful methods to assist further use of viability sets produced by the computation.

Keywords: Viability theory, kd-tree, decision support

1. Introduction

Mathematical Viability Theory, as stated by Aubin (1991), is a mathematical framework designed to study the compatibility between evolutions of dynamical systems and constraints in the state space. A viability problem is generally defined with a controlled dynamical system S, (which may be a multi-valued map), a set of admissible controls U, (which may depend on the state of the system) and a set of constraints, K, which is a subset of the state space. The viability kernel $viab_S(K)$ is a subset of the constraints

*Corresponding author
Email addresses: isabelle.alvarez@irstea.fr (Isabelle Alvarez), romain.reuillon@iscpif.fr (Romain Reuillon), ricardo.de.aldama@gmail.com (Ricardo de Aldama)

set K that gathers all the states from which it is possible to maintain at least one evolution within K. Its capture basin gathers the states from which it is possible to reach the viability kernel. More generally it is possible to compute the capture basin of a target in the constraint set, as done for example by Tinka et al. (2009). The viability kernel classifies the states into *viable* and *not viable*. Once the viability kernel is computed, it is possible to use the information generated by the viability analysis. Viability analysis and the sets it produces (the viability kernel and its capture basin) can be used to define, test and compare control strategies (Sabatier et al., 2015; Gourguet et al., 2015; Andrès-Domenech et al., 2014; Rapaport et al., 2006), to simulate particular viable evolutions or to optimize objective function inside the viability kernel or capture basin (Sicard et al., 2012; Mesmoudi et al., 2010), to illustrate concepts related to sustainability and trade-off (Martinet and Doyen, 2007; Hardy et al., 2013; Wei et al., 2013), or concepts related to resilience, robustness and risk (Martin, 2004; Alvarez and Martin, 2011; Accatino et al., 2014; Rougé et al., 2013). Aubin et al. (2011) gives more examples.

In some cases it is is possible to provide an analytical description of a viability kernel (see the consumption model by Aubin (1991), the population model by Aubin and Saint-Pierre (2007), and the bilingualism model by Bernard and Martin (2012)). But generally it is necessary to compute approximations. The main algorithm for computing exact viability kernel was established by Saint-Pierre (1994), it proposes a numerical scheme that approaches the viability kernel by a decreasing sequence of sets that contain the viability kernel (possibly empty). This algorithm has been used as a basis in numerous problems (Aubin et al., 2011)). It is a numerical scheme with proved convergence towards the viability kernel in the most general conditions (infinite horizon). However, it is difficult to operate since the problem dynamic is embedded in the algorithm for convergence and efficiency reasons. It is also memory consuming since viable states are represented on a regular grid. In the case of capture basin, several algorithms are available, and in some cases it is possible to compute both an inner and outer approximation (Lhommeau et al., 2011). But for the computation of the viability kernel with infinite horizon, there is still no bound of convergence. To bypass these difficulties, other algorithms restrict their objective to the computation of an approximation at a given horizon (as done by Maidens et al. (2013), with a very efficient Lagrangian method for Linear models, or like Doyen and Lara (2010) with dynamic programming in stochastic case) or the computation of a set of trajectories (as done by Bonneuil (2006) with simulated annealing to generate trajectories near the boundary). An alternative to Saint-Pierre's algorithm, using classification functions instead of the regular grid, is proposed by Deffuant et al. (2007). When the classification function verifies some specific conditions, the convergence towards the viability kernel is guaranteed. Although very attractive, in particular to handle multidimensional control space, the Support Vector Machine (SVM) functions used by Chapel and Deffuant (2011), don't fulfill the conditions of the convergence theorem.

Besides, since obtaining a viability kernel is a difficult and time consuming task, it is often considered as a result in itself, and the potential information provided by the viability kernel is not fully used. Moreover, there are only a few control strategies available in the general case to govern the system in the viability kernel at each time step, since it can be very difficult to prove that a particular choice of control builds an

evolution. These are mainly the *heavy* (or *inertia*) strategy by Aubin (1991) (which consists in minimizing the norm of the control rate of change at each time step), and the *slow* strategy by Falcone and Saint-Pierre (1987), which consists in minimizing the norm of the control vector (at each time step). In practice both strategies are very difficult to operate since the trajectories they produces can spend a lot of time on the boundary of the viability kernel (Alvarez and Martin, 2011). Since the viability kernel is approximated by its outside this is a robustness issue: Lying on the boundary of the approximation can mean being outside the true viability kernel.

In any case the computation of the viability kernel can be seen as a preliminary step in the study of the governance of an evolutionary system subjected to state constraints. Once the viability kernel (and possibly its capture basin) is computed, it is possible to compare decisions, scenarios, or control strategies from the viability viewpoint. It is also possible to perform a viability analysis, to answer to sustainable management issues: is it possible to maintain the desirable properties in the future ? And if so, which actions are required ? If not, is it possible to restore these properties and how ? It is also possible to take into account new criteria which were not available prior to the viability study, particularly the robustness to perturbation in the state space (considering perturbation as uncertainty in a similar way to Regan et al. (2005)). This approach was successfully followed in a real application of Camembert cheese ripening process (Mesmoudi et al., 2014). The viability study showed that it was possible to reach the target (a *good* Camembert cheese) in 8 days rather than 12. Within the viable set it was possible to select a sequence of actions that was tested in a pilot and proved to be both viable and robust. However, the computation of the viable sets necessary to the study was very difficult to implement (it was a 5 dimensions problem) and it was not possible to develop reusable components. The present work was undertaken to facilitate viability studies, which require the computation of several viability kernels and basic tools to exploit the viability results.

In this paper we propose a framework in order to compute and study viability kernels and capture basins when the dynamical system is described by a set of differential equations. Its objective is to propose basic tools and output functions in order to consider viability kernels as tools for decision support. The computation algorithm uses classification functions and active learning as proposed by Deffuant et al. (2007). It is designed to be as independent as possible on the differential equations, so that the framework can be used for different types of viability problems and also to give the possibility to improve or modify the viability algorithm easily.

As classification functions we adapted kd-trees (proposed by Bentley (1975) to store a set of points in a p-dimensional space) to represent a set with finer divisions near its boundary. This choice of classification function preserves the set-theoretic vision of viability kernels (since basic operations on this structure of kd-trees can be implemented without learning stage). It is very easy to implement a set of constraints in this framework, without being limited to hyper-rectangles or simple polytopes, since the same kd-tree structure can be used to approximate the constraint set. This kd-tree structure generally compresses the information stored on the regular grid, although in the worst case it is less efficient (computing an empty viability kernel requires the complete splitting of the search space, see Rouquier et al. (2015) for more details).

This article is organized as follows: In the next section we present the viability

problem, and we recall the approximation algorithm with classification function. In section 3, we present our kd-tree structure, and we show how it can be used to approximate a set. We describe our viability approximation algorithm with kd-tree as learning functions. We also present the basic functions we provide in order to conduct a viability analysis. In section 4 we presents examples that show how to define viability problems in our framework, together with computation of viability kernel and comparison with theoretic viability kernels. We present a viability study and examples of viability-based decision issues. Limits and possible improvements are discussed in this section. The main conclusions of this work are summarized in section 5.

2. Viability problem: Approximation with classification functions

We consider a viability problem defined by a controlled dynamical system S, a set-valued map U (the set of admissible controls depending on the state of the system), and a subset K of the state space (the set of constraints):

$$(S) \left\{ \begin{array}{lcl} x'(t) & = & \Phi(x(t), u(t)) \\ u(t) & \in & U(x(t)) \end{array} \right. \tag{1}$$

where $x(t)$ is the state of the system S, $x(t) \in X$ a finite dimensional vector space ; $u(t)$ is the control, with $u(t) \in Y$ a finite dimensional vector space (in the following we consider $X = \mathbb{R}^p$) and $Y = \mathbb{R}^q$). The set-valued map $U : X \rightsquigarrow Y$ gives the set of admissible control for each state $x \in X$. Φ is a function from Graph(U) to X.

Let $K \subset X$ a compact subset of X be the set of desirable states, the constraint set in which the state $x(t)$ is supposed to stay. The viability kernel $viab_S(K)$ is the subset of K (possibly empty) that gathers the states from which it is possible to find a control function $u(t)$ such that the evolution $x(.)$ stays in the compact set K.

$$x \in viab_S(K) \Leftrightarrow \exists u(.) \quad \forall t \geq 0 \left\{ \begin{array}{lcl} x'(t) & = & \Phi(x(t), u(t)) \\ u(t) & \in & U(x(t)) \\ x(t) & \in & K \end{array} \right. \tag{2}$$

Following the method described in Deffuant et al. (2007), the dynamical system S is discretized in time. We consider the time step $dt > 0$, and we considered the discrete controlled dynamical system (S_{dt}) defined by the set-valued map $F : X \rightsquigarrow X$

$$F(x) \quad = \quad \{x + \Phi(x, u)dt, u \in U(x)\} . \tag{3}$$

We assume that F is μ-Lipschitz with closed images, which means that the images of two states x and y can't diverge from more than $\mu \|x - y\|$: $\forall x, \forall y \in K, F(y) \subset F(x) + \mu \|x - y\| B$ where B is the unit ball, and F(x) is a closed set.

We consider the viability kernel $viab_{S_{dt}}(K)$ of the discretized dynamical system (S_{dt}) with constraint set K. The viability theory theorems from Aubin (1991) state that $viab_{S_{dt}}(K)$ is the largest subset Z of K verifying: $\forall x \in Z, F(x) \bigcap Z \neq \emptyset$.

We consider a discrete grid K_h defined on K such that:

$$\forall x \in K, \exists x_h \in K_h, \text{ such that } \|x - x_h\| \leq \beta(h) \tag{4}$$

where β is a function defined on $\mathbb{R}^+$ such that $\beta(h) \to 0$ when $h \to 0$. There exists such grids since K is a compact set.

Let us consider a learning algorithm LA that uses as input a learning set of pairs $(x_i, e_i) \in K \times \{0,1\}$ and produces a characteristic function g of a connected set Γ as output. The approximation algorithm builds a decreasing sequence of discrete subsets K^n of K_h, $K^{n+1} \subset K^n \subset K_h$, and a sequence of characteristic functions g^n built with the learning algorithm LA from the learning sets $K^n \times \{1\}$ and $K_h \backslash K^n \times \{0\}$.

We note $L(K^n)$ the inverse image of 1 by g^n: $L(K^n) = (g^n)^{-1}(1) = \{x \in K, g^n(x) = 1\})$. The initialization of the sequence is the following: $K^0 := K_h$.
The recursive definition of the sequence is the following:

$$K^{n+1} = \{x_h \in K^n \mid d(F(x_h), L(K^n)) \leq \mu\beta(h)\}\,. \tag{5}$$

This sequence converges in finite steps towards a set K'_h possibly empty (since the sequence is decreasing). The approximation theorem from Deffuant et al. (2007) states that $L(K'_h)$ converges towards $viab_{S_{dt}}(K)$ under some conditions.

Theorem 1 (fact from Deffuant et al. (2007))**.** *The set produced by the learning algorithm $L(K'_h)$ converges towards $viab_{S_{dt}}(K)$ when h tends to 0 if the following conditions are fulfilled:*

$$L(K^0) = K \tag{6}$$

$\exists \lambda > 0, \quad \forall n \geq 0$

$$\begin{cases} \forall x \in L(K^n) \quad d(x, K^n) \leq \lambda\beta(h) & (7) \\ \forall x \in K \backslash L(K^n) \quad d(x, K_h \backslash K^n) \leq \beta(h) & (8) \end{cases}$$

The learning algorithm kd-LA we use here relies on a data structure based on kd-trees which is presented in section 3.1 (and fully described by Rouquier et al. (2015)). It verifies the conditions of Theorem 1, (see 3.3) so when the spatial discretization step h tends to 0 the kd-tree approximation tends to the viability kernel.

It uses as input a characteristic function $\mathbb{1}_S$ where S is a subset of a hyperrectangle of $\mathbb{R}^p$ (for example, with a learning set of of pairs $(x_i, e_i) \in K \times \{0,1\}$, we would have $S = \{x_i \texttt{ with } e_i = 1\}$, with $S \subset K \subset R$, where R is a hyperrectangle that bounds K. There exists such a hyperrectangle since K is a compact set. The associated learning function t provides as a result $t(\mathbb{1}_S)$, an approximation of $\mathbb{1}_S$ built on a tree structure G, refined from root to leaves. Each node covers the hyperrectangle covered by the leaves it contains. The root node of G is defined on R. Each leaf has a label, either 0 or 1. So if we note $\{l_i\}$ the set of leaves of G, a state $x \in R$ belongs to one leaf only l_x, so $t(x) = label(l_x)$ and $t = \mathbb{1}_{G_1}$, where $G_1 = \{l_i,\ label(l_i) = 1\}$. When K is a hyperrectangle of $\mathbb{R}^p$ (or more generally the set G_1 of a kd-tree structure G), the kd-tree based learning function t is such that $t^0 = t(K_h, \emptyset) = \mathbb{1}_K$. In other cases $\mathbb{1}_K$ is directly used instead of t^0 in the initialization step of the viability algorithm.

3. Algorithms

The approximation method with learning function implies the computation of $L(K^n)$ at each step n with the learning algorithm kd-LA. Besides, in order to handle a great class of constraint sets, and not only hyperrectangles, it is useful to propose an algorithm to approximate a compact set included in a hyperrectangle $R \subset \mathbb{R}^p$. This is the reason why we first design a learning algorithm from an oracle with kd-trees, as described by Rouquier et al. (2015).

3.1. Learning a Set with kd-trees: The kd-LA algorithm

We consider that a function $f : R \subset \mathbb{R}^p \mapsto \{0, 1\}$ is available, where f is the indicator $\mathbb{1}_S$ of a compact simply connected set S subset of the hyperrectangle R. This function is called the *oracle*. Calls to the *oracle* can be very costly depending on the set S, but they can be easily parallelized.

We consider the general learning algorithm, that builds a kd-tree, approximation of a given set S with discretization step $h > 0$ as in (4), using the indicator function $f = \mathbb{1}_S$ as oracle (see Rouquier et al. (2015) for a complete description).

Algorithm 1. ***LearnBoundary(***$root, h, \mathbb{1}_S$***)*** *kd-LA algorithm*

```
root is the leaf tree corresponding to R (the area to explore),
or a kd-tree, with at least a positive leaf
```

0. $G \leftarrow root$
1. $A \leftarrow$ `leavesToRefine`(G, h) *(a set of distinct non atomic leaves)*
2. `while` $A \neq \emptyset$ `do {`
3. `foreach` $node_i \in A$ `do {`
4. `Refine(`$node_i, \mathbb{1}_S, \alpha(h)$`)}`
6. $A \leftarrow$ `leavesToRefine`(G, h)
7. `return` G

`Refine`$(node, \mathbb{1}_S, \alpha(h))$ creates two children at node $node$ if the stopping criterion $\alpha(h)$ is not reached. When the stopping criteria is reached the corresponding leaf is called atomic. The stopping criterion defines the maximal length of the path of a node or the minimal size of the area associated with the node.

The discretization step of the grid h in equation (4) is linked to the stopping criteria α depending on the method that is used to divide a node, since several methods can be used: splitting it along its largest dimension if the length of the path from the root is smaller than α; splitting it following a cyclic order (while the number of division is lower than α), etc. The default method we used is the following: a leaf is no longer divided if the length of its path to the root node is greater than $\alpha = p\alpha'$ where p is the dimension of the state space, and a leaf is always divided along its largest dimension. In these conditions we have $h = \frac{1}{2^{\alpha'/p}}$, and $h \to 0$ when $\alpha \to \infty$.

If $node$ is represented by the vector of intervals $([a_i, b_i[)_{1,..,p}$ and is divided along j, its children will have the same vector values except for the j^{th} coordinate which is $[a_j, \frac{(b_j - a_j)}{2}[$ for one child and $[\frac{(b_j - a_j)}{2}, b_j[$ (if $b_j \neq 1$) for the other ($[\frac{(1 - a_j)}{2}, 1]$ with $b_j = 1$). Several methods can be considered to assign a label to the children. Using the center of the leaf requires two calls to the oracle to label the two children. Here we

systematically reuse the point of the parent node to label the child it belongs. A point is randomly drawn to label the other leaf and aligned on the closest point of the $h = \frac{1}{2^{\alpha/p}}$ grid. The image of this grid point by the oracle gives the label of the leaf. It is possible (as a parameter of the algorithm) to disregard the grid alignment and use directly the randomly drawn point.

At each step, all the refining operations involve different leaves of the tree, so the calls to the oracle in order to label the new leaves can be parallelized. This is an option of Algorithm 1. Algorithm `leavesToRefine`(G, h) recursively gathers positive border leaves (that is leaves that are on the boundary of the exploratory set with positive label) and pairs of adjacent leaves with different labels.

Leaves that are refined are non atomic leaves next to the boundary of the hyperrectangle R (the area to explore), and non atomic adjacent leaves with different labels. Two leaves are adjacent when they share a boundary: When leaves are described by an interval on each axis, for each axis, the interval of one leaf is included in (or equal to) the other, except for exactly one coordinate axis where both intervals are adjacent (their closures have exactly one common point). Since there is only a finite number of atomic leaves in the hyperrectangle R, and in the worst case all leaves are eventually atomic, Algorithm `leavesToRefine` returns an empty set in finite time. So algorithm (1) `BuildG` terminates in finite time.

When the learning set is a close set (respectively an open set), atomic leaves adjacent to a leaf with different label have their interval definition to be changed in order to include (respectively exclude) the adjacent boundary.

3.2. Operations on kd-trees

In order to provide the sequence of sets defined by equation (5), and to provide analysis tools, we propose some operators on the kd-trees.

Let G be a kd-tree produced with Algorithm 1. The `dilation` algorithm computes a basic dilation of G. Every atomic leaves of the boundary are labeled to 1 and the tree is refined again. This operation is repeated p times (where p is the dimension of the state space), in order to perform an approximation of the Euclidean dilation.

Algorithm 2. ***Dilation(***$G = \mathbb{1}_S, h, \nu \in \mathbb{N}$***)***
`returns a kd-tree` G_ν `with adjacent atomic leaves in` G `labeled to` 1 `recursively` ν `times`
0. $i \leftarrow 0$, $G_i \leftarrow G$
1. `while` $i \neq \nu$ `do {`
2. $leaves \leftarrow \{$`boundaryLeaves`$(G_i)\}$
3. `foreach` $l \in leaves$ `do` $\{$`label`$(l) \leftarrow 1\}$
4. $G_{i+1} \leftarrow$ `LearnBoundary`$(G_i, h, \mathbb{1}_{G_i})$
5. $i \leftarrow i + 1\}$
7. `return` G_ν

`boundaryLeaves` gathers pairs of adjacent atomic leaves with different labels (we call these pairs critical pairs). We note $S_{<\nu>}$ the indicator set for $G_\nu =$ `Dilation` (G, h, ν) (that is ν basic dilation steps), and $S_{<p\nu>} = S_\nu$ the indicator set for $G_{p\nu}$ ($p\nu$ basic dilation steps corresponding to ν standard dilation steps when p is the dimension of the space). We note $\oplus$ the Minkowski sum of subsets in an Euclidean space:

$A \oplus B = \{a + b \mid a \in A,\ b \in B\}$. In the following we note $A \oplus r$ the Minkowski sum of set A and the Euclidean ball of radius r. With the dilation algorithm, we have:

Theorem 2. *A $\nu \in \mathbb{N}$ steps standard dilation ($p\nu$ basic dilation) verifies:*

$$S \oplus \nu h \subset S_\nu \subset S \oplus \nu hp$$

The proof is given in AppendixA.

Let G be a kd-tree such that $G = \mathbb{1}_S$. By switching the label of the leaves of G (label 1 becomes 0 and conversely), we define another kd-tree $G' = \mathbb{1}_{S'}$ with $S' = K \backslash S$. We consider now the `erosion` algorithm, which consists basically in labeling every atomic leaves of the boundary to 0 and refining the tree again. We have by construction, when S and its dilation are subsets of the interior of K:

$$(\texttt{Erosion}(G, h, 1))' = \texttt{Dilation}(G', h, 1)$$

If we note $S_{<-1>}$ the indicator set of the kd-tree built by the `Erosion` algorithm, we have:

$$K \backslash S_{<-1>} = (K \backslash S)_{<1>} \tag{9}$$

We note $\ominus$ the Euclidean erosion operator: $S \ominus B(\rho) = \{x \in S,\ B(x, \rho) \subset S\}$. We note here $S \ominus \rho = S \ominus B(\rho)$, and we note $S_{-\nu}$ the set produced by $\nu \in \mathbb{N}$ steps standard erosion ($p\nu$ basic erosion). From Theorem 2 and from (9), we have:

$$K \backslash S \oplus \nu h \subset (K \backslash S)_\nu \subset K \backslash S \oplus \nu hp$$

Corollary 1. *A $\nu \in \mathbb{N}$ steps standard erosion ($p\nu$ basic erosion) verifies:*

$$S \ominus \nu hp \subset S_{-\nu} \subset S \ominus \nu h$$

Proof. We already have $K \backslash S \oplus \nu h \subset K \backslash S - \nu \subset K \backslash S \oplus \nu hp$. We now recall that with $\rho > 0$, we have: $(K \backslash S) \oplus \rho = K \backslash (S \ominus \rho)$. Let $x \in K \backslash (S \ominus \rho)$, if $x \in S$ then there is $y \in B(x, \rho)$ such that $y \notin S$, so $x = y + (x - y) \in (K \backslash S) \oplus \rho$. Conversely, if $x \in (K \backslash S)$ and $s \in B(\rho)$, if $x + s \in S$, then $(x + s) - s = x \notin S$ so $x + s \in K \backslash (S \ominus \rho)$. We then have: $K \backslash (S \ominus \nu h) \subset K \backslash S - \nu \subset K \backslash (S \ominus \nu hp)$. ∎

With the `erosion` algorithm, it is necessary to specify which parts of the boundary are to be eroded, since it depends on the motives of the user (an example is given in section 4.1.4). In the kd-tree viability framework, the end-user defines a specific domain. The parts of the boundary that reach the domain boundary are not to be eroded.

3.3. Computation of the Viability Kernel with kd-LA

The viability kernel is computed with a kd-tree that stores at each leaf:

- the input grid point x associated to the leaf (`test point`)
- the `label` of the leaf

- the `control index` i of a viable control $u_i \in U(x)$ if the label is true
- the `result point` $F(x, u_i)$ of x by the dynamics (3)
- the index of the next control to test in the discretized set $\texttt{Dis}(U(x))$ in the case of the exhaustive search of a viable control ($1+$ `argmax` $\{\ i | \forall j \leq i,\ F(x, u_j)$ is not viable$\}$).

The algorithm follows the general guidelines of section 2 with the definition of the sequence K^n in equation (5). Calls to the oracle are aligned on the grid such that $\beta(h) = \frac{h}{2}\sqrt{p}$.

Algorithm 3. ***ComputeVK(***$root$***,***h***,***F***,***$\mathbb{1}_K$***,***μ***)***
$\mathbb{1}_K$ *is an indicator function of the constraint set* K
1. `Define` $Oracle(x) = 1$ `iff` $\exists u \in \texttt{Dis}(U(x)),\ F(x, u) \in K,\ 0$ *otherwise*
2. $G_0 \leftarrow$ `buildInitialTree`$(SearchArea, h, Oracle)$
3. with `Define function viabilityStep`(G_n)
4. *if* $(G_n = \emptyset)$ `return` $\emptyset$
5. *else* $G_{n+1} \leftarrow$ `buildStepVK`(G_n, h, F, μ)
6. *if* $(G_n = G_{n+1})$ `return` G_n *else* `return viabilityStep`(G_{n+1})
7. `return viabilityStep`(G_0)

SearchArea is a hyperrectangle R which contains the constraint set $K \subset R$. The definition of the Oracle in algorithm (3) is the initialization step of the sequence G_n. **buildInitialTree** returns, if it succeeds, a tree which contains one leaf for which the label is true. Otherwise it returns the empty set: This means that the viability kernel is empty (for this specific value of h), the algorithm couldn't find a single atomic leaf for which the image of the grid point stays in Kat the first time step. Algorithm (4) defines the sequence G_n with the updated Oracle following equation (5).

Algorithm 4. ***buildStepVK(***G_n***,***h***,***F***,***μ***)***
1. $G_\mu \leftarrow$ `Dilation`$(G_n, h, \nu(\mu))$
2. `Define Oracle` $\leftarrow \{Oracle(x) = G_\mu(F(x))\}$
3. $G_{n+1} \leftarrow$ `findTrueLabel(Reassign(`G_n`),Oracle)`
4. `If` $G_{n+1} = \emptyset$ `Then return` $\emptyset$
5. `Else return learnBoundary(`G_{n+1}`,`h`,Oracle)`

The function ν determines from the Lipschitz parameter of F the input of the basic dilation that must be perform in order to include all points at a distance $\mu\frac{h}{2}\sqrt{p}$. (From Theorem 2, we have $\nu(\mu) = p(E(\frac{\mu}{2}\sqrt{p}) + 1)$ when $\frac{\mu}{2}\sqrt{p} \notin \mathbb{N}$, and $\frac{\mu}{2}p\sqrt{p}$ otherwise).

`Reassign` creates a clone of the tree in which leaves with label 0 are preserved while leaves with label 1 are updated: It checks whether their result point is still viable with the new oracle, using first the previously registered control value.

Algorithm 5. ***findTrueLabel(***$tree$***,***$Oracle$***)***
1. `If` $(\exists leaf \in tree;$ `label`$(leaf) = 1)$ `Then return` $tree$
2. `Else refineBiggestLeaves`$(tree)$
3. with `Define function refineBiggestLeaves`$(tree)$
4. $leaves \leftarrow \{leaf$ `with shortest path from the root` $s \neq h\}$

5. `If` $(leaves = \emptyset)$ `Then return` $\emptyset$
6. `Else`
7. `For each` $leaf \in leaves$ `Refine`$(leaf, Oracle)$
8. `If` $(\exists leaf \in leaves; \texttt{label}(leaf) = 1)$ `Then return` $tree$
9. `Else refineBiggestLeaves`$(tree)$

`findTrueLabel` and `buildInitialTree` differ mainly by the way of choosing the control values to be tested (since in `buildInitialTree` no previous control is available).

Implementation examples and indication to user are available on the site [1]

With slight modification of Algorithm 3 it is possible to compute the capture basin of a target in a search area. The search area D is bounded by a hyperrectangle R and defined by an indicator function. The target set T is first learned (if it is not already a kd-tree) with Algorithm 1 (`LearnBoundary`). Then a time step algorithm `buildStepBC` is applied. It is similar to Algorithm 4 (`buildStepVK`), except for the `Reassign` step in which leaves with label 1 (already in the capture basin) are preserved and leaves with label 0 are updated. All control values are tested for this update (since a control that did not previously drive a point in the capture basin can be successful at a following time step).

3.4. Convergence

In order to build a close set, it is necessary to assign a close interval in the direction of adjacency to boundary leaves with label 1. In that case, Algorithm `ComputeVK` produces a kd-tree G which is an outer approximation of the viability kernel $viab_{S_{dt}}(K)$ as defined in Section 2, when some smoothness and regularities conditions are fulfilled.

Let us consider as particular learning function L the function that associates to $x_h \in K_h$ the closed ball of the sup-norm of radius $h/2$ centered on x_h: $L_t(x_h) = B_\infty(x_h, h/2)$. We note M^n the sequence built from definition (5) with $L = L_t$, $M^0 = K_h$.

Proposition 1. *L_t verifies the conditions (7) and (8) of Theorem 1.*

The proof is given in AppendixB.

We consider the following set of hypotheses H1, depending on F and K only:
The sequence M^n is such that there exists for each $n \in \mathbb{N}$, a closed set $M(n) \subset K$ verifying $M^n \subset M(n)$ and $K_h \backslash M^n \subset K \backslash M(n)$, and the sets $M(n)$ verify: $\exists \rho_1 > 0$, $\exists \rho_2 > 0$, $\forall n \in \mathbb{N}$,

$$\begin{cases} \forall x \in M(n),\ \exists y_x \in M(n),\ x \in B(y_x, \rho_1) \subset M(n) & (10) \\ \forall x \in K \backslash M(n),\ \exists y_x \in K \backslash M(n),\ x \in B(y_x, \rho_2) \subset K \backslash M(n) & (11) \\ M(n) \ominus B(\rho_1) \texttt{ and } (K \backslash M(n)) \ominus B(\rho_2) \texttt{ are path-connected} & (12) \end{cases}$$

The set of hypotheses H1 ensures that the viability kernel has no tentacles possibly as thin as wanted, that can't be discovered by the learning algorithm. Let us consider with

[1] https://github.com/ISCPIF/viabilitree

the notation of Section 2, the kd-tree $G^n = L(K^n)$ built at step n with Algorithm (4) `BuildStepVK`.

Proposition 2. *When hypothesis H1 is verified, the sequence K^n defined by the kd-tree learning algorithm L is the sequence M^n defined by L_t.*

Proposition 3. *When the initialization condition $L(K^0) = K$ is fulfilled and H1 is verified, the set produced by the kd-tree learning algorithm converges towards $viab_{S_{dt}}(K)$ when h tends to 0.*

This last proposition is the consequence of Propositions 1 and 2. Proof of the propositions is given in AppendixB.

4. Application and Discussion

4.1. Comparison with Exact Solution

Very few analytical solutions of viability kernel are available. We used the examples from Aubin (1991), Aubin and Saint-Pierre (2007), and Bernard and Martin (2012).

4.1.1. Population growth model

This example is taken from Aubin and Saint-Pierre (2007). The population model is defined from Maltus and Verhulst (Verhulst, 1845). The population viability problem consists in maintaining the size of the population in a given interval $[a; b]$. The state of the system is described by the variables $x(t)$, the size of the population, and $y(t)$, the population growth rate. The dynamics are described by the following equations:

$$\left\{ \begin{array}{lcl} x(t+dt) & = & x(t) + x(t)y(t)dt \\ y(t+dt) & = & y(t) + u(t)dt \ with \ |u(t)| \leq c \end{array} \right. \tag{13}$$

The dynamics are controlled by taking the growth rate evolution in interval $[-c, c]$. This viability problem can be resolved analytically (see Aubin and Saint-Pierre (2007) for details). When dt tends toward 0, the theoretical viability kernel is defined by:

$$Viab(K) = \left\{ (x, y) \in \mathbb{R}^2 |\ x \in [a; b], y \in [-\sqrt{2c\log(\frac{x}{a})}; \sqrt{2c\log(\frac{b}{x})}] \right\} \tag{14}$$

Figure 1 shows approximations obtained with the kd-tree viability algorithm. When the dilation is applied there is some diffusion but it is an outer approximation. Table 1 gives a summary of the accuracy of the approximation for approximation with dilation parameter $\nu = 0$ and $\nu = 1$. The approximation without dilation ($\nu = 0$) is already a good approximation (only 13 grid points are false negative). This is due to the Lipschitz constant of the dynamics $x' = c.x$ which is $|c|$-Lipschitz continuous. Here we have $c = 0.5$.

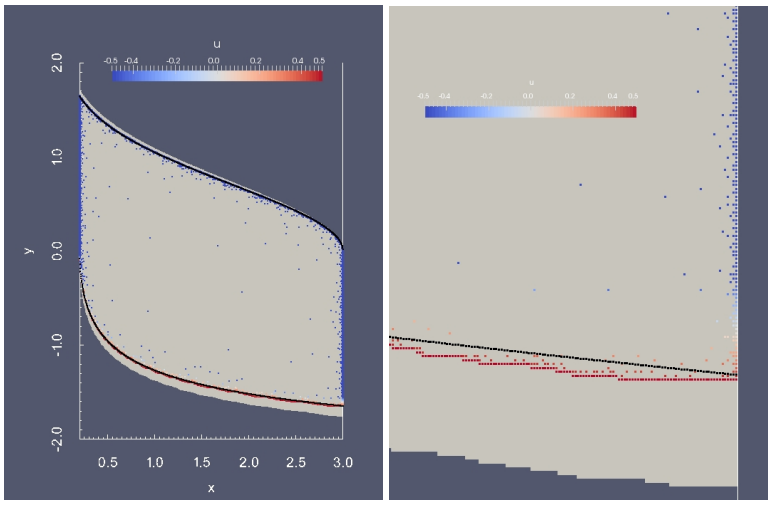

Figure 1: Approximation of the viability kernel with 1024 points / axis for the population problem with constraint set $K = [a = 0.2, b = 3] \times [d = -2, e = 2]$, parameters $dt = 0.1$ (integration step 0.01), control set $U = [-0.5; 0.5]$ with discretization step 0.02. In gray the approximated viability kernel with dilation parameter $\nu = 1$. Colored points are testing points of the leaves of the approximation with $\nu = 0$. The color stands for the value of the control u. In black the boundary of the true kernel. Detailed view on the right.

Dilation ν	Approximation size	FP size	FP rate	FN size	FN rate
0	571 238	4 695	8.218991e-03	13	2.294618e-05
1	605 330	38 774	6.405432e-02	0	0

Table 1: Accuracy summary for the population viability problem, with the parameters of figure 1. The size of the sets are in number of points of the grid. The size of the true viability kernel is 566 556 points. False Positive (FP) are points of the approximation that are not in the true viability kernel $viab(K)$. The FP rate is compared to the size of the approximation (number of positive points). False Negative (FN) are points of the true kernel that are not in the approximation. The FN rate is compared to the size of the true positive points.

4.1.2. Consumption Model

The consumption model is proposed by Aubin (1991) to describe the consumption of raw material governed by price. The state variable $x(t)$ represents the consumption of the raw material, and the state variable $y(t)$ its price. The rate of change at each time step of the price is controlled and bounded par parameter c with $u(t) \in [-c, c]$. The constraint set is $K = [0, b] \times [0, d]$. The dynamics are described by the following equations:

$$\begin{cases} x(t+dt) &= x(t) + (x(t) - y(t))dt \\ y(t+dt) &= y(t) + u(t)dt \; with \; |u(t)| \leq c \end{cases} \tag{15}$$

This viability problem can be resolved analytically (see Aubin (1991) for details). When dt tends toward 0, the theoretical viability kernel is defined by:

$$((x,y) \in [0,b] \times [0,d]) \in Viab(K) \Leftrightarrow \begin{cases} x \geq y - c + c.e^{(-y/c)} \\ \text{and when } y \leq b \text{ then } x \leq y + c - c.e^{\frac{y-b}{c}} \end{cases} \quad (16)$$

The corresponding dynamics in dimension 1 is $x' = x - c$, it is Lipschitz continuous with constant $\mu = 1$. As it can be seen on Figure 2, the approximation with no dilation

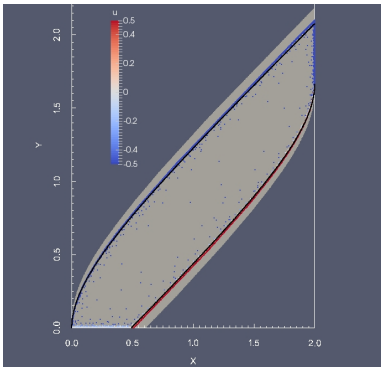

Figure 2: Approximation of the viability kernel for the consumption problem with parameters $b = 2$, $d = 3$, $c = 0.5$, depth=20 (1024 points per axis), $dt = 0.05$ (integration step 0.001), 0.1. for the control discretization step. In gray the viability kernel with basic dilation parameter $\nu = 2$. Colored points are the testing points of the leaves of the approximation with $\nu = 0$. The color stands for the value of the control u. In black color points of the boundary of the true kernel

Dilation ν	Approximation size	FP size	FP rate	FN size	FN rate
0	296 109	11 031	3.73e-02	1	3.51e-06
2	350 292	65 213	1.86e-01	0	0

Table 2: Accuracy summary for the consumer viability problem, with the parameters of figure 2. The size of the sets are in number of points of the grid. The size of the true viability kernel $viab(K)$ is 285 079 points. False Positive (FP) are points of the approximation that are not in $viab(K)$. The FP rate is compared to the size of the approximation (number of positive points). False Negative (FN) are points of the true kernel that are not in the approximation. The FN rate is compared to the size of the true positive points. ν is the number of basic dilation.

0 ($\nu = 0$) is almost an outer approximation. Table 2 gives a summary of the accuracy of the approximation with parameters of Figure 2. The $\nu = 2$ approximation is an outer approximation of the true viability kernel, although the diffusion is rather important.

4.1.3. Bilingualism model

This example is taken from Bernard and Martin (2012). The bilingual society model is based on the work from Abrams and Strogatz (2003) which studies language

competition. It considers as state variables the proportions of monolingual speakers of each language (σ_A and σ_B, with the proportion of bilingual speakers $1-\sigma_A-\sigma_B$), and the prestige s of language A (as $1-s$ is the prestige of language B). The value of the prestige can evolve with public action, so $\frac{ds}{dt}$ is considered as a control on the dynamics, bounded in some interval $U=[-\bar{u};\bar{u}]$ with $\bar{u}>0$.

$$\left\{\begin{array}{lcl} \frac{d\sigma_A}{dt} & = & (1-\sigma_A-\sigma_B)(1-\sigma_B)^a s-\sigma_A\sigma_B^a(1-s) \\ \frac{d\sigma_B}{dt} & = & (1-\sigma_A-\sigma_B)(1-\sigma_A)^a(1-s)-\sigma_B\sigma_A^a s \\ \frac{ds}{dt} & = & u\in U \end{array}\right. \tag{17}$$

If s is constant in $]0;1[$, the dynamics has three equilibria: $(0,1)$, $(1,0)$ which are stable, and an unstable one. Consequently, one language is doomed to become extinct. Therefore it is necessary to apply control policy on s to insure the coexistence. The viability domain is described by Bernard and Martin (2012), and an approximation was computed for the following parameters and set of desirable set in $E=[0,1]^3$, $K=\{(\sigma_A,\sigma_B,s)\}$ such that:

$$\left\{\begin{array}{rcl|l} 0<\underline{\sigma}\leq & \sigma_A & \leq 1 & \underline{\sigma}\text{=}0.2 \\ 0<\underline{\sigma}\leq & \sigma_B & \leq 1 & \bar{u}\text{=}0.1 \\ 0\leq & s & \leq 1 & a\text{=}1.31 \end{array}\right. \tag{18}$$

Parameter a is set according to the literature. (It is calibrated in Abrams and Strogatz (2003) from historical data). The domain D was computed by labeling hypercubes of edge size $1/100$ when they contain at least a point of the viability kernel.

Figure 3 shows the 3D-view of the viability domain D and the shape of approximation obtained with the kd-tree viability algorithm. Figure 4 shows in a sliced 2D-view the difference between the domain D and the kernel approximation obtained with the kd-tree viability algorithm for several value of the dilation and time step parameters. It is not possible to disregard the Lipschitz coefficient for the bilingual model, since the viability kernel approximation with $\nu=0$ is not an outer approximation for $dt=1$.

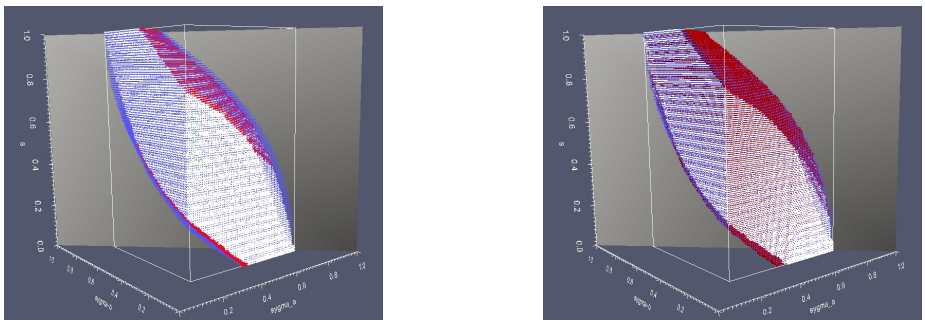

Figure 3: 3D view of the bilingual domain with parameters from (18), discretization step of 0.01 for the control. In blue the kernel computed by labeling hypercubes of size $1/100$ when they contain at least a point of the viability domain D. The viability kernel obtained with the kd-tree LA are computed with $depth=21$, i.e. $2^7=128$ points per axis, in white with $\nu=0$, in red with $\nu=1$. On the left with $dt=1$, (with integration step 0.1), on the right $dt=0.5$, (with integration step 0.05)

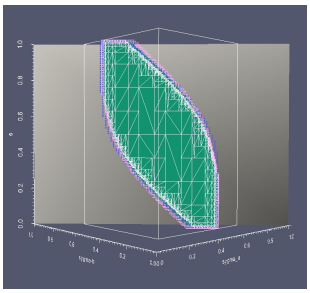

Figure 4: Measurement of ν effect. A sliced view of the bilingual domain (plane H defined by its normal vector (110) and a point $(0.420.420.5)$). In blue the kernel computed by labeling hypercubes of size $1/100$ when they contain a point of the viability kernel. The approximation are computed with $depth = 21$, i.e. $2^7 = 128$ points per axis, in dark green with $\nu = 1$, in light green with $\nu = 2$ and pink with $\nu = 3$ with time step 1 and integration step 0.1.

Figure 5 shows in sliced 2D-views the influence of time discretization (controlled with parameter dt), which is important.

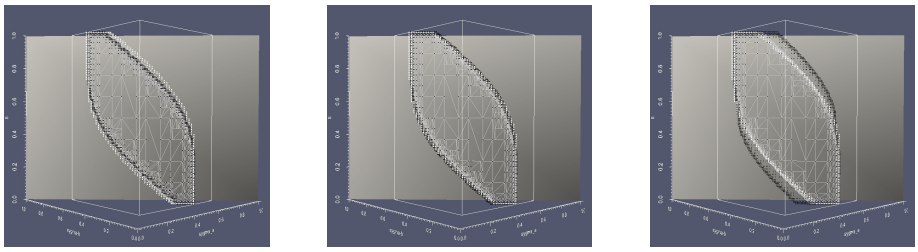

Figure 5: Measurement of the time effect. Sliced views of the bilingual domain (plane H defined by its normal vector (110) and a point $(0.420.420.5)$). In white the boundary points of the viability kernel. The approximation are computed with $depth = 21$, i.e. $2^7 = 128$ points per axis, in black with $dt = 0.5$, in light grey with $dt = 1$. From left to right with $\nu = 0$, $\nu = 1$ and $\nu = 3$.

Table 3 gives a summary of the accuracy of the approximation with $\nu = 0$ and $\nu = 1$. As it can be seen on Figure 5, the approximation without dilation ($\nu = 0$) is completely inside the viability domain. With $dt = 0.5$ the approximation is better and with ν increasing the approximation more points of the true domain.

4.1.4. Lake and nearby farms

The lake and nearby farms viability problem is based on Martin (2004). The issue is to determine how a lake can remain in a clear-water (oligotrophic) state and not shift to turbid-water (eutrophic) state while agricultural activity is performed. The problem can be modeled by the dynamic between phosphorus concentration P and phosphorus inputs L (excess phosphorus due to fertilizer and animal feed supplements input through leaching and runoff). Agriculture requires a minimum value for L, $Lmin$, and oligotrophic state imposes a maximum value for P, $Pmax$. Regulation laws are constraints on $\left|\frac{dL}{dt}\right| \leq V$: this means that the regulation law cannot set the maximum amount of phosphorus inputs, but rather imposes a decrease of the phosphorus inputs,

Time step	Dilation ν	Approx. size	FP size	FP rate	FN size	FN rate
0.5	0	332 026	0	0	44 534	1.34e-01
0.5	1	374 094	2 628	7.03e-03	5 094	1.37e-02
0.5	3	441 291	66 521	0.15	1 790	4.78e-03

Table 3: Accuracy summary for the languages viability problem, with the parameters of figure 3. The size of the sets are in number of points of the grid. The size of the true viability domain (once translated on the more refined grid) is 376 560 points. False Positive (FP) are points of the approximation that are not in the true viability domain $viab(K)$. The FP rate is compared to the size of the approximation (number of positive points). False Negative (FN) are points of the true kernel that are not in the approximation. The FN rate is compared to the size of the true positive points.

for example by a percentage each year. This decrease is bounded by V. The viability problem is the following:

$$\begin{cases} (L,P) & \in & K = [L_{min}, L_{max}] \times [0, P_{max}] \\ U & = & [-V, V] \\ \frac{dL}{dt} & = & u(t) \in U \\ \frac{dP}{dt} & = & -bP(t) + L(t) + r\frac{P^q(t)}{m+P^q(t)} \end{cases} \tag{19}$$

Parameters b,m,r,q depends on the lake and in particular they express the possible hysteresis and irreversibility of the eutrophication (see Carpenter et al. (1999) for details), since the equilibrium locus presents a cusp in the (L, P, b) space. The viability kernel is bounded on the left by the line $L = L_{min}$. On the right the theoretical boundary can be computed by identifying the trajectory that reaches the intersection between the equilibrium locus and the boundary of K (with maximum value of P), while applying the most constraining control ($u = -V$).

Figure 6 shows approximations obtained with the kd-tree viability algorithm. When the dilation is applied this is an outer approximation. When the dilation parameter is 0 the approximation can be an outer parameter depending on the grid step. Table 4 gives a summary of the accuracy of the approximation for approximation with $\nu = 0$ and $\nu = 1$ with depth = 20 and 16 respectively. As it can be seen on Figure 6, the approximation without dilation ($\nu = 0$) can already contain the viability domain depending on the depth parameter. The accuracy of the approximation is far better with the diminution of the spatial discretization step (corresponding to increasing depth parameter).

4.2. Viability-based analysis

Dealing with sustainable issues in environmental sciences problems but not only, classical utility maximization approach are challenged by constraint-based approach such as the Safe Minimum Standard (Ciriacy-Wantrup, 1952) or Tolerable Windows Approach (TWA) (Petschel-Held et al., 1999). As it was already shown by Bruckner et al. (2003) for TWA, or by Martinet and Doyen (2007), the mathematical viability theory offers a natural framework to study the compatibility between constraints and dynamical systems. To define and solve a viability problem requires the definition by

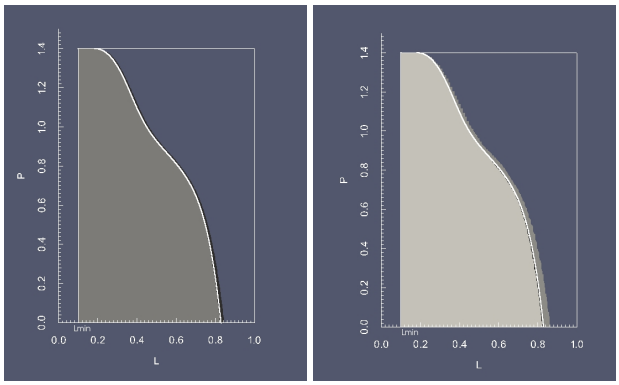

Figure 6: Approximation of the viability kernel for the lake problem with parameters $m = r = 1$, $q = 8$, $b = 0.8$, $V = 0.9$, $dt = 0.1$ (integration step 0.01), discretization step for the control is 0.1. On the left different values of ν for $depth = 20$ (1024 points per axis). In dark (light) gray the viability kernel with dilation parameter $\nu = 1$ ($\nu = 0$). In white the outline of the constraint set $K = [0.1; 1] \times [0; 1.4]$. In white color points of the boundary of the true kernel. On the right the approximation for different value of the $depth$ parameter (with $\nu = 0$). In light gray $depth = 18$, in color black $depth = 20$ (very close to the true kernel), in color dark gray $depth = 16$.

Depth	Dilation ν	Approximation size	FP size	FP rate	FN size	FN rate
20	0	587 346	3	5.107722e-06	3 340	5.69e-03
20	1	605 986	15 303	0.025	0	0
16	0	38 675	1 665	0.043	0	0
16	1	43 047	6 124	0.142	0	0

Table 4: Accuracy summary for the lake viability problem, with the parameters of figure 6. The size of the sets are in number of points of the grid. The size of the true viability kernel $viab(K)$ is 590 683 points with depth=20 (1024 points per axis) and 36 923 with depth=16 (256 points per axis). False Positive (FP) are points of the approximation that are not in $viab(K)$. The FP size is compared to the size of the approximation (number of positive points). False Negative (FN) are points of the true kernel that are not in the approximation. FN size is compared to the size of the true positive points (approximation - FP size).

the stakeholders of the set of desirable states (the constraint set K) and of admissible control set U. It requires also sufficient knowledge on the dynamical system, including the effect of control. Before considering control strategy or action scenarios, it is necessary to agree on desirable states, possible controls and dynamics, which allow to define a viability problem (S).

The computation of the viability kernel is then the consequence of the definition of the viability problem (S). Once the viability kernel $viab_S(K)$ is known, it can be used for several viability analysis and decision support.

In particular, when the viability kernel is empty, it means that the definition of the constraints set and the policy are inappropriate regarding the dynamics. Before any consideration of optimization and thus of objective function, it is essential for the stakeholders to discuss the definition of desirable states and the means to achieve it

(here the set of admissible controls). When the viability kernel is not an empty set, is is useful to examine the sensitivity of the viability kernel to the parameters of the viability problem (constraint set, control set, dynamics). The present framework can be useful to compare different version of the viability problems because of its modularity. Once the viability problem is well stated, it is possible to consider further issues, such as the choice of the control strategy, robustness or resilience of the system to perturbations.

4.2.1. Choice of a control strategy

As an example we rely on the case of the lake and the lakeside farms (see section 4.1.4). The viability kernel is the subset of the (L, P)-plane that gathers all states (L, P) such that there exists at least one regulation law that allows the oligotrophic state to be maintained. Figure 7 shows the viability kernel in the case of a reversible lake, with the value of the viable control associated to the test point of each leaf. A leaf

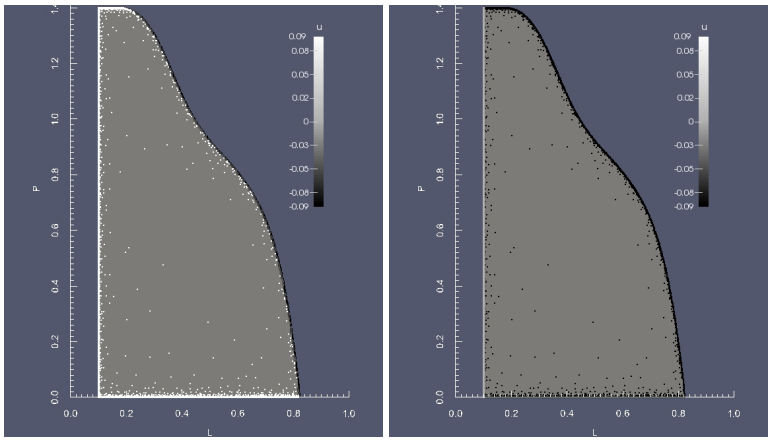

Figure 7: Viable control value for test points in the viability kernel (in light grey) for the lake problem with parameters of Figure 6 and $\nu = 0$. On the left controls are ordered from positive values to negative ones. On the right controls are tested in reverse order.

is declared viable as soon as one control value maintains the test point in the current version of the viability kernel. This means that only one control value is recorded for each viable leaf, although there may be several alternatives. Actually, inside the interior of the viability kernel, in continuous time, every controls are viable (although it can be for a very short time only). On the boundary, on the contrary, not all the controls are viable. Figure 7 shows in particular that near $L = L_{min}$, the control value can't be negative (the point color is no longer black). Near the boundary with maximum values of L, the control value gets closer to -0.09 and reaches it on the boundary: the point color goes from white to black when the distance to the boundary tends to 0.

Modifying the order in which the control values are tested manifests which constraints apply on the control near the boundary.

The order is set by the user and is state-dependent, so it is possible to apply a cost function depending from the state in order to get straight in one pass for each state the viable control that minimizes locally the control cost.

In the lake problem, the cost of the control for the farmers is obviously increasing with the control value, so the simplest definition of the control set that respects the order $\succ$ induced by this cost function is not state-dependent: $u_1 \succ u_2 \Leftrightarrow u_1 > u_2$. Figure 7 shows the result of order (left view). More complicated criteria could easily be defined for optimization purpose.

4.2.2. Defining a new decision criteria: the robustness to perturbation in the state space

In many applications, perturbations can occur in the state space, with the result of possible shift of the state of the system. Uncertainties about the state of the system can also be seen as its belonging to a set rather than a single vector. For these reasons, it is interesting to consider the distance to the boundary of the viability kernel. Another reason is technical: The viability kernel is approximated by its outside, which means that near the boundary there is a risk to be already outside the true viability kernel, and thus to be doomed to leave the constraint set in the future. Once the viability kernel has been computed, it is possible to use its boundary to define robustness criteria as in Alvarez and Martin (2011).

In the kd-tree viability framework we propose to use the `erosion` algorithm in order to define a new constraint set, which is an eroded set from the viability kernel. Let (S) be a viability problem defined by equation 2, and let $H = \texttt{viab}_S(K)$ be the viability kernel. We assume that $H \neq \emptyset$. We consider the eroded set $H_{\ominus\nu}$ computed from H with the erosion algorithm: $H_{\ominus\nu}$ is the indicator set of the kd-tree returned by $\texttt{erosion}(\mathbb{1}_H, h, \nu)$. We then define a new viability problem:

$$x \in viab_S(H_{\ominus\nu}) \Leftrightarrow \exists u(.) \quad \forall t \geq 0 \begin{cases} x'(t) & = & \Phi(x(t), u(t)) \\ u(t) & \in & U_{\ominus}(x(t)) \\ x(t) & \in & H_{\ominus\nu} \end{cases} \tag{20}$$

This new viability kernel $\texttt{viab}_S(H_{\ominus\nu})$ gathers the states in $H = \texttt{viab}_S(K)$ that are at least at a given distance $h\nu$ from the boundary of $\texttt{viab}_S(K)$ and from which there is a trajectory that stays that far from the boundary with controls in $U_{\ominus}(x)$. We call these trajectories conservative trajectories, since they stays at distance from the boundary of the original viability kernel.

For example, Figure 8 shows the viability kernel $viab(H)$ obtained with 40 steps erosion of the viability kernel from figure 7, with the same control than for the original viability problem and with a less costly subset of the original control set (assuming that in dangerous situation it is possible to ask for greater efforts than in everyday situation). Trajectories in $\texttt{viab}_S(H_{\ominus\nu})$ are robust to a perturbation in the state space (with size smaller than the erosion size $h\nu$): the state will be shifted possibly outside $\texttt{viab}_S(H_{\ominus\nu})$ but still inside $H = \texttt{viab}_S(K)$, which means that after the perturbation it will be possible to find a control strategy to stay in H. This was not possible in H. A small perturbation of a state near the boundary shifts the state outside H, which means that the lake is doomed to become eutrophe (to leave K) in the future, at least for a while.

This robustness criterion can be considered in optimization objectives or as an independent decision criteria when exploring the viability kernel for optimal solutions or comparing scenarios.

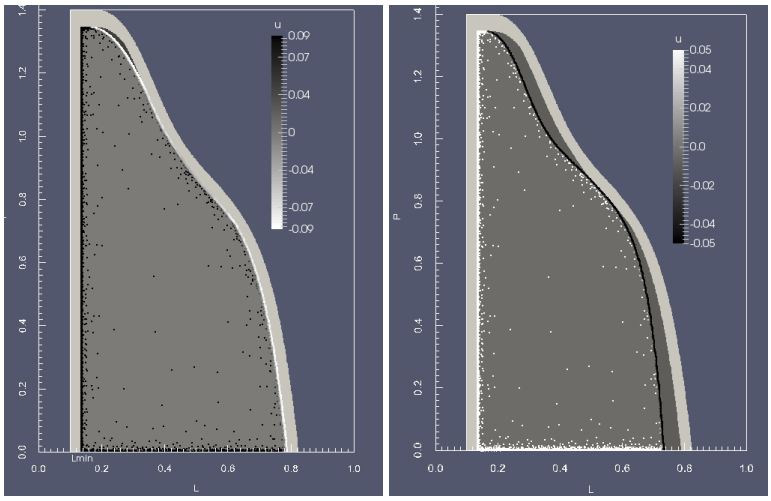

Figure 8: Viable kernel (in grey) of the 40 steps eroded viability kernel (in dark grey) defined with the parameters of figure 7. The original viability kernel (defined from K) is in light grey. On the left with the same state of admissible controls. On the right with $V = 0.05$ and points colored according to their viable control value.

Other criteria can be considered such as the resilience value (as defined in Martin (2004)), using the capture basin algorithm. It was done by Alvarez et al. (2013) for the model of language competition.

4.3. Limits and possible improvement of the framework

Viability algorithm. In order to guarantee the convergence towards the true viability kernel, it is necessary to consider as viable, points that are in a dilated set within a distance that depends on the Lipschitz constant of the dynamic. In the present version the algorithm uses an upper bound of the Lipschitz constant, although it could be more efficient to estimate the Lipschitz constant locally as done by Saint-Pierre (1994).

Dynamic. The present computation of the image of a state by the discretized dynamics is done with a fixed time step, which is inefficient when the system exhibits both slow and rapid dynamics. When the discretization step in time is too small relatively to the spatial discretization step, slow local dynamics can be seen as invariant and the classification function cannot be updated. Finding an efficient global discretization step in space and time is difficult. It would be preferable to handle the time step locally.

Controls. In the present framework, heuristic methods can be implemented in order to find a viable control, but the size of the set of admissible control is assumed to be constant. Further work includes variable size of the control set and the implementation of local-guided search (to propose candidates for adjacent leaves). Besides it could be also interesting to compute the complete set of viable controls $\widehat{U}(x_l) \subseteq U(x_l)$ at x_l, or at least part of it (for instance, a given number of viable controls). It is expensive in

computation time but can be very useful for optimization issues. The modular architecture of the framework allows to perform these changes easily through the content stored at each leaf.

Cleaning leaves. In the current implementation, leaves close to the true boundary are maximally refined. Since the same kd-tree is used again at different time steps, when the dynamic is fast the final kd-tree can be maximally refined in many unnecessary regions. This is memory consuming and inefficient. It would be preferable to clean the kd-tree on a regular basis, by merging pair of leaves of the same parent with identical label. Cleaning operation requires technical choice to merge information stored at the leaf level (test points, viable control, etc.)

Distance. The distance of a state to the boundary of the viability kernel can be used to characterize the robustness to perturbation, not only for a given state but also for trajectories (as it is proposed by Alvarez and Martin (2011)). Presently it is possible, using dilation and erosion functions, to propose distance-based strategy with the distance induced by the kd-tree. But the present erosion function needs some improvement: the definition of the boundaries that are not to be eroded by the erosion function is limited to subset of hyperplanes; and the rosion algorithm will gain in efficiency with systematic cleaning. The computation of other distance could be also very useful. We are currently working in this direction by extending basic operations on the trees.

Optimization. Once the viability kernel is computed, it is possible to save it (with the `save` function) in order to explore it later for further use. Optimization criteria could be easily implemented, if several viable controls are stored for each viable leaf. This would enable the end-user to test different control scenarios and to perform intertemporal optimization (with the help of external optimization modules).

Sampling strategy. Currently the test points are taken from a regular grid. This was convenient to fulfill the conditions of the convergence theorem. Random sampling is also already implemented. It could be interesting to study the impact of different sampling strategies on the approximation result.

Curse of dimensionality. As it is the case with many learning algorithms, the present method is very sensitive to the dimension of the state and control spaces. However, parallelization could be used more extensively to bypass this problem. Presently only the sampling of the test points is done in parallel. The modular structure of the framework offers many other parallelization opportunities for the test of images over several time steps, the dilation and cleaning process. Parallelizations through graphics processing units could also be considered as it as be done by for discrete viability algorithm by Brias et al. (2016).

5. Conclusion

We have presented a general framework in which viability-based decision problems can be easily implemented and analyzed. The core of the framework is an algorithm

that computes the viability kernel, using kd-trees as underlying data structure. The approximation is computed recursively. The dynamics equations are used to define a suitable indicator function to guarantee the convergence. We prove that the algorithm fulfills the conditions of the convergence theorem of the approximation algorithm with classification function. We also compare the resulting viability kernels with available analytical solutions. We have shown an example of what can be achieved in terms of decision support for the problem of the lake and nearby farms. Our approach is focused on modularity and re-usability: The algorithms are implemented in Scala and are available in a free and open-source implementation [2]. On the site additional information is available to help the user to design new viability problems and contribute to the implementation. This is a possibility for external validation and further development of viability studies and tools for decision support. The main drawback of the framework is its limit with state dimension. Further work is in progress to improve efficiency with parallelization and to extend the viability-based decision support framework with distance and robustness operators.

Acknowledgements

This work was partly supported by the European Community's Seventh Framework Programme under the grant agreement no. FP7-222 654-DREAM.

6. References

Abrams, D. and Strogatz, S. (2003). Modelling the dynamics of language death. *Nature*, 424(6951):900.

Accatino, F., Sabatier, R., De Michele, C., Ward, D., Wiegand, K., and Meyer, K. M. (2014). Robustness and management adaptability in tropical rangelands: a viability-based assessment under the non-equilibrium paradigm. *Animal*, 8:1272–1281.

Alvarez, I., de Aldama, R., Martin, S., and Reuillon, R. (2013). Assessing the Resilience of Socio-Ecosystems: Coupling Viability Theory and Active Learning with kd-Trees. In *proceedings of IJCAI*, pages 2776–2782.

Alvarez, I. and Martin, S. (2011). Geometric robustness of viability kernels and resilience basins. In Deffuant, G. and Gilbert, N., editors, *Viability and Resilience of Complex Systems*, Understanding complex systems, pages 193–218. Springer.

Andrès-Domenech, P., Saint-Pierre, P., Fanokoa, P. S., and Zaccour, G. (2014). Sustainability of the dry forest in androy: A viability analysis. *Ecological Economics*, 104:33 – 49.

Aubin, J. and Saint-Pierre, P. (2007). An introduction to viability theory and management of renewable resources. In Kropp, J. and Scheffran, J., editors, *Advanced*

[2] https://github.com/ISCPIF/viabilitree

Methods for Decision Making and Risk Management, pages 43–80. Nova Science Publishers.

Aubin, J.-P. (1991). *Viability theory*. Birkhäuser, Basel.

Aubin, J.-P., Bayen, A., and Saint-Pierre, P. (2011). *Viability Theory: New Directions*. Springer.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Bernard, C. and Martin, S. (2012). Building strategies to ensure language coexistence in presence of bilingualism. *Appl.Math.Comput.*, 218(17):8825–8841.

Bonneuil, N. (2006). Computing the viability kernel in large state dimension. *Journal of Mathematical Analysis and Applications*, 323(2):1444 – 1454.

Brias, A., Mathias, J.-D., and Deffuant, G. (2016). Accelerating viability kernel computation with cuda architecture: application to bycatch fishery management. *Computational Management Science*, pages 1–21.

Bruckner, T., Petschel-Held, G., Leimbach, M., and Toth, F. L. (2003). Methodological aspects of the tolerable windows approach. *Climatic Change*, 56:73–89.

Carpenter, S. R., Ludwig, D., and Brock, W. A. (1999). Management of eutrophication for lakes subject to potentially irreversible change. *Ecol Appl.*, 9:751–771.

Chapel, L. and Deffuant, G. (2011). Approximating viability kernels and resilience values: Algorithms and practical issues illustrated with kaviar software. In Deffuant, G. and Gilbert, N., editors, *Viability and Resilience of Complex Systems*, pages 161–192. Springer.

Ciriacy-Wantrup, S. (1952). *Resource Conservation: Economics and Policies*. University of California Press, Berkeley.

Deffuant, G., Chapel, L., and Martin, S. (2007). Approximating viability kernels with support vector machines. *IEEE T. Automat. Contr.*, 52(5):933–937.

Doyen, L. and Lara, M. D. (2010). Stochastic viability and dynamic programming. *Systems & Control Letters*, 59(10):629 – 634.

Falcone, M. and Saint-Pierre, P. (1987). Slow and quasi-slow solutions of differential inclusions. *Nonlinear Analysis: Theory, Methods & Applications*, 11(3):367–377.

Gourguet, S., Thébaud, O., Jennings, S., Little, L., Dichmont, C., Pascoe, S., Deng, R., and Doyen, L. (2015). The cost of co-viability in the australian northern prawn fishery. *Environmental Modeling & Assessment*, pages 1–19.

Hardy, P.-Y., Béné, C., Doyen, L., and Schwarz, A.-M. (2013). Food security versus environment conservation: A case study of solomon islands' small-scale fisheries. *Environmental Development*, 8:38 – 56.

Lhommeau, M., Jaulin, L., and Hardouin, L. (2011). Capture basin approximation using interval analysis. *International Journal of Adaptive Control and Signal Processing*, 25(3):264–272.

Maidens, J. N., Kaynama, S., Mitchell, I. M., Oishi, M. M., and Dumont, G. A. (2013). Lagrangian methods for approximating the viability kernel in high-dimensional systems. *Automatica*, 49(7):2017 – 2029.

Martin, S. (2004). The cost of restoration as a way of defining resilience: a viability approach applied to a model of lake eutrophication. *Ecol. Soc.*, 2(9):8.

Martinet, V. and Doyen, L. (2007). Sustainability of an economy with an exhaustible resource : A viable control approach. *Resour. Energy Econ.*, 29(1):17–39.

Mesmoudi, S., Alvarez, I., Martin, S., Reuillon, R., Sicard, M., and Perrot, N. (2014). Coupling geometric analysis and viability theory for system exploration: Application to a living food system. *Journal of Process Control*, 24(12):18–28.

Mesmoudi, S., Perrot, N., Reuillon, R., Bourgine, P., and Lutton, E. (2010). Optimal viable path search for a cheese ripening process using a multi-objective ea. In *Proc. of Int. Conf. on Evolutionary Computation (ICEC)*, pages 225–230.

Petschel-Held, G., Schellnhuber, H.-J., Bruckner, T., Toth, F., and Hasselmann, K. (1999). The tolerable windows approach: Theoretical and methodological foundations. *Climatic Change*, 41(3-4):303–331.

Rapaport, A., Terreaux, J., and Doyen, L. (2006). Viability analysis for the sustainable management of renewable resources. *Math Comput Model.*, 43(5,6):466–484.

Regan, H., Ben-Haim, Y., Langford, B., Wilson, W., and Lundgerg, P. (2005). Robust decision-making under severe uncertainty for conservation management. *Ecological Applications*, 15(4):1471–1477.

Rougé, C., Mathias, J., and Deffuant, G. (2013). Extending the viability theory framework of resilience to uncertain dynamics, and application to lake eutrophication. *Ecological Indicators*, 29:420–433.

Rouquier, J.-B., Alvarez, I., Reuillon, R., and Wuillemin, P.-H. (2015). A kd-tree algorithm to discover the boundary of a black box hypervolume. *Annals of Mathematics and Artificial Intelligence*, pages 1–16.

Sabatier, R., Oates, L., and Jackson, R. (2015). Management flexibility of a grassland agroecosystem: A modeling approach based on viability theory. *Agricultural Systems*, 139:76 – 81.

Saint-Pierre, P. (1994). Approximation of the viability kernel. *Applied Mathematics & Optimisation*, 29(2):187–209.

Sicard, M., Perrot, N., Reuillon, R., Mesmoudi, S., Alvarez, I., and Martin, S. (2012). A viability approach to control food processes: Application to a camembert cheese ripening process. *Food Control*, 23(2):312 – 319.

Tinka, A., Diemer, S., Madureira, L., Marques, E., Sousa, J., Martins, R., Pinto, J., Silva, J., Sousa, A., Saint-Pierre, P., and Bayen, A. (2009). Viability-based computation of spatially constrained minimum time trajectories for an autonomous underwater vehicle: Implementation and experiments. In *Proc. the 2009 American Control Conference*, pages 3603–3610.

Verhulst, P.-F. (1845). Recherches mathématiques sur la loi d'accroissement de la population. *Nouveaux Mémoires de l'Académie Royale des Sciences et Belles-Lettres de Bruxelles*, 18:1–42.

Wei, W., Alvarez, I., and Martin, S. (2013). Sustainability analysis: Viability concepts to consider transient and asymptotical dynamics in socio-ecological tourism-based systems. *Ecological Modelling*, 251:103 – 113.

AppendixA. Proof of theorem 2

Let h be the size of the grid (it is also the size of the kd-tree smallest leaf). Let p be the dimension of the space. We note A^C the complement of set A in the state space. We consider a kd-tree G built with algorithm 1, such that $G = \mathbb{1}S$. We note $S_{<1>}$ the indicator set for G_1 (one basic dilation step), and $S_{<p>} = S_1$ the indicator set for G_p (p basic dilation steps corresponding to one standard dilation step in dimension p).

We recall Theorem 2: A $\nu \in \mathbb{N}$ steps standard dilation ($p\nu$ basic dilations) verifies:

$$S \oplus \nu h \subset S_\nu \subset S \oplus \nu hp$$

Proof. We first prove that if $y \in S_{<1>}$, we have: $d(y, S) \leq h$.

If $y \notin S$, y belongs to a former labeled 0 leaf from a pair of atomic leaves with different labels. This critical pair has a unique common border. The orthogonal projection of y on this common border is in S. Since the size of an atomic leaf is h, then we have $d(y, S) \leq h$.

We do a standard dilation step (p basic dilation steps) and let $y \in S_{<p>}$, we have: $d(y, S) \leq ph$. So $S_{<p>} \subset S \oplus ph$. So a $\nu \in \mathbb{N}$ steps standard dilation verifies:

$$S_\nu \subset S \oplus \nu hp \tag{A.1}$$

Conversely, let us consider a standard dilation step (p basic dilation steps). We show that $d((S_{<p>})^C, S) \geq h$.

Proof. Let $z \notin S$ such that $d(z, S) < h$. Let z' be a point of S where the distance is reached, and L' the leaf to which z' belongs. z' belongs to the boundary of L' (since the distance is reached for z'). L' is represented by a vector of intervals. Let $A(z, z') \neq \emptyset$ be the set of coordinates i for which $z_i - z'_i \neq 0$.

We note $\sharp A$ the cardinal of set A. If $\sharp A = 1$ with $A = \{j\}$ this means that the leaf L defined with the same intervals as L' for $i \neq j$ and with the adjacent interval for the jth coordinate (in z direction) is labeled 0. L is necessary atomic since it is adjacent to a labeled 1 leaf, so $\{L, L'\}$ is a pair of critical leaves and $z \in L$ since $d(z, S) \leq h$. With the definition of the dilation algorithm, L will be labeled to 1 at the first basic dilation step, so $L \subset S_{<1>}$ and $z \in S_{<1>}$.

We now show by induction the following property. Let $z \notin S,\ d(z, S) < h$. Then:

$$\exists z' \in S,\ d(z, z') < h \text{ and } \sharp A(z, z') = k \leq p \text{ then } z \in S_{<k>}. \tag{A.2}$$

We first prove that this property is true for $k = 1$. Let $z' \in S,\ d(z, z') < h$. As above, there is some $v' = z' + t(z - z')$ with $t \in [0, 1)$ such that $d(z, v') \leq d(z, z') < h$ and v' belongs to the boundary of a labeled 1 leaf L', adjacent to the labeled 0 leaf to which z belongs. So $z \in S_{<1>}$.

Let us assume that this proposition A.2 is verified for $k < p$. We now consider the case where $\sharp A = k + 1$. Without any generality loss we can assume that $1 \in A$. Let L' be the leaf to which z' belongs. Let us consider $v = z' + (z - z').e_1$, where $e_i, i \leq k+1$ are the unit vectors of the orthonormal basis. We have $d(z, v) < d(z, z') < h$, and $A(z, v) = A(z, z') - 1$. If $v \in S$, then $v \in S_{<1>}$. Proposition A.2 applies to z with $v \in S_{<1>}$, so $z \in (S_{<1>})_{<k>} = S_{<k+1>}$. Otherwise, if $v \notin S$, let us consider the

leaf L defined with the same intervals as L' for coordinates $j \neq i = 1$ and with the adjacent interval for the first coordinate (in $z - z'$ direction). v belongs to L with label 0. $\{L, L'\}$ is a pair of critical leaves, so L will be labeled to 1 at the first basic dilation step, so $L \subset S_{<1>}$ and $v \in S_{<1>}$. So we can apply the proposition to z and $v \in S_{<1>}$ at step k: we have $z \in (S_{<1>})_{<k>} = S_{<k+1>}$.

So if $z \notin S$ such that $d(z, S) < h$, then $z \in S_{<p>}$, and so $d((S_{<p>})^C, S) \geq h$ ■

Since $d((S_{<p>})^C, S) \geq h$, we have $S_{<p>} \supset S \oplus h$. If we perform a $\nu \in \mathbb{N}$ steps standard dilation, we then have:

$$S_\nu \supset S \oplus \nu h \tag{A.3}$$

With equations A.1 and A.3, we have shown that : $S \oplus \nu h \subset S_\nu \subset S \oplus \nu p h$ □

AppendixB. Proof of Proposition 3

We note $\ominus$ the erosion operator: $S \ominus B(\rho) = \{x \in S,\ B(x, \rho) \subset S\}$.

We recall the definition of the sequence built by the learning-algorithm L. The initialization of the sequence is: $K^0 := K_h$.
The recursive definition of the sequence is the following:

$$K^{n+1} = \{x_h \in K^n \mid d(F(x_h), L(K^n)) \leq \mu\beta(h)\}. \quad (5)$$

The conditions of the convergence theorem for learning functions are the following:

$$L(K^0) = K \quad (6)$$

$$\exists \lambda > 0,\ \ \forall n \geq 0 \quad \begin{cases} \forall x \in L(K^n) \quad d(x, K^n) \leq \lambda\beta(h) & (7) \\ \forall x \in K \backslash L(K^n) \quad d(x, K_h \backslash K^n) \leq \beta(h) & (8) \end{cases}$$

We now prove that the kd-tree algorithm verify the conditions of Theorem 1.

Proof. We consider the learning function L_t that associates to a vector x_h of the grid the ball of the sup-norm of radius $h/2$ centered on x_h: $L_t(x_h) = B_\infty(x_h, h/2)$. We note M^n the sequence built from definition 5 with $L = L_t$, $M^0 = K_h$. We first prove Proposition 1.

Proposition (1)**.** *L_t verifies the conditions (7) and (8) of Theorem 1.*

Proof. Let $x \in L_t(M^n)$, then by construction there exists $x_h \in M^n$ such that $x \in B_\infty(x_h, h/2)$, so condition 7 is verified. Let $x \in K \backslash L_t(M^n)$, and let $x_h \in K_h$ such that $d(x, x_h) \leq \beta(h)$. We have $x \in B_\infty(x_h, h/2)$, so necessarily $x_h \notin M^n$ and condition 8 is verified. ■

Let us consider a kd-tree $G^n = \mathbb{1}_{S^n}$ built at step n with Algorithm (4). If the kd-tree learning algorithm uses at each step all the vectors from the sequence M^n, then $K^n = M^n$ and $S^n = L_t(M^n)$. But the main interest of the method is to use only a subset of the grid. The kd-tree learning algorithm uses a subset H^n of M^n and a subset I^n of $K_h \backslash M^n$ to compute $L(K^n)$ and when not all leaves are atomic, H^n and I^n are strictly included in M^n and $K_h \backslash M^n$ respectively. K^n denotes here the grid vectors

that are labeled to 1 as a result of the learning algorithm. In order to show that the kd-tree learning algorithm verifies the conditions (7) and (8) of Theorem 1, we prove that $K^n = M^n$.

We recall the set of hypotheses H1, depending on F and K only:
The sequence M^n is such that there exists for each $n \in \mathbb{N}$, a closed set $M(n) \subset K$ verifying $M^n \subset M(n)$ and $K_h \backslash M^n \subset K \backslash M(n)$, and the sets $M(n)$ verify: $\exists \rho_1 > 0$, $\exists \rho_2 > 0$, $\forall n \in \mathbb{N}$,

$$\begin{cases} \forall x \in M(n),\ \exists y_x \in M(n),\ x \in B(y_x, \rho_1) \subset M(n) & (10) \\ \forall x \in K \backslash M(n),\ \exists y_x \in K \backslash M(n),\ x \in B(y_x, \rho_2) \subset K \backslash M(n) & (11) \\ M(n) \ominus B(\rho_1) \texttt{ and } (K \backslash M(n)) \ominus B(\rho_2) \texttt{ are path-connected} & (12) \end{cases}$$

We now prove Proposition 2: When hypothesis H1 is verified, the sequence K^n defined by the kd-tree learning algorithm L is the sequence M^n defined by L_t.

Proof. Let $x_h \in K_h \backslash K^n$. We note l the leaf of G^n to which x_h belongs. The label of l is 0. If the label of l is evaluated at x_h, then $x_h \in I^n \subset K_h \backslash M^n$. We now prove that $x_h \in K_h \backslash M^n$ when $x_h \notin I^n$.

Let us suppose that $x_h \in M^n$. Then from hypothesis H1.10 there is $x' \in M(n) \ominus B(\rho)$ such that $d(x_h, x') \leq \rho$. Let $y_h \in H^n$. There is such a vector since $S^n \neq \emptyset$. (Otherwise, if $S^n = \emptyset$ the kd-tree is completely refined, and in that case we have $x_h \in I^n$). Since $H^n \subset M^n$, with Hypothesis (10), there is $y' \in M(n) \ominus B(\rho)$ such that $d(y_h, y') \leq \rho$. With Hypothesis (12) there is a path $\gamma_2 : [t_1, t_2] \mapsto M(n) \ominus B(\rho)$ connecting x' and y' in $M(n) \ominus B(\rho)$, a segment $\gamma_3 : [t_2, 1] \mapsto B(y', \rho)$ connecting y' and y_h, and a segment $\gamma_1 : [0, t_1] \mapsto B(x', \rho)$ connecting x' and x_h. We call γ the path from x_h to y_h built from $\gamma_1,\ \gamma_2,\ \gamma_3$. The label of y_h is 1 while the label of x_h is 0 (since it is the label of l), so the boundary of $L(K^n)$ is crossed along γ: there is a pair of critical leaves (z, z') along the path, with $z_h \in z \cap I^n$ and $z'_h \in z' \cap H^n$, and $0 \leq t_1 < t_2 \leq 1$ with $\gamma(t_1) \in z$ and $\gamma(t_2) \in z'$ We note $t_1 < T \leq t_2$ such that $\gamma(T) \in (\bar{z} \cap \bar{z'})$. The boundary cannot be crossed along γ_2 (since $h << \rho$, we would have $z \subset M(n)$ and consequently $z_h \in M^n$, which is not possible since $z_h \in I^n$). For the same reason it cannot be crossed along γ_3, since $y_h \in H^n$, and even if $y_h = z'_h$ we have $d(z_h, y') < d(y_h, y')$ when h is small enough compared to ρ. (Since $\gamma(T)$, y_h, z_h, y' belongs to the same plane P and in that plane, the projection of $\gamma(T)$ on the segment $[y_h, z_h]$ is $\frac{y_h + z_h}{2}$). This implies that $z_h \in B(y', \rho)$ and therefore $z_h \in M^n$, which is impossible. The same reasoning shows that it cannot be along γ_1 either: since $x_h \notin I^n$, $x_h \neq z_h$ and $x_h \neq z'_h$, so $d(z'_h, x') < d(x_h, x')$, so $z' \subset M(n)$, which is not possible. So $x_h \in K_h \backslash M^n$.

We then have $K_h \backslash K^n \subset K_h \backslash M^n$. Since symmetrically, we have $K^n \subset M^n$, then $K^n = M^n$. ■

So if H1 is verified, the sequence K^n built by the kd-tree learning algorithm is $K^n = M^n$ (with Proposition 2). This sequence verifies the conditions (7) and (8) of Theorem 1 (with Proposition 1). So if $L(K^0) = K$, Theorem 1 applies and the set produced by the kd-tree learning algorithm converges towards $viab_{S_{dt}}(K)$ when h tends to 0. □

The authors thank JE Dabas for the petanque-like proof.