

# Reconstruction of Epsilon-Machines in Predictive Frameworks and Decisional States

Nicolas Brodu <nicolas.brodu@univ-rennes1.fr>\*

May 2011

## Abstract

This article introduces both a new algorithm for reconstructing epsilon-machines from data, as well as the *decisional states*. These are defined as the internal states of a system that lead to the same decision, based on a user-provided utility or pay-off function. The utility function encodes some a priori knowledge external to the system, it quantifies how bad it is to make mistakes. The intrinsic underlying structure of the system is modeled by an epsilon-machine and its causal states. The decisional states form a partition of the lower-level causal states that is defined according to the higher-level user's knowledge. In a complex systems perspective, the decisional states are thus the "emerging" patterns corresponding to the utility function. The transitions between these decisional states correspond to events that lead to a change of decision. The new REMAPF algorithm estimates both the epsilon-machine and the decisional states from data. Application examples are given for hidden model reconstruction, cellular automata filtering, and edge detection in images.

*Keywords:*  $\epsilon$ -machines; decisional states; utility; predictions.

## 1 Motivation

We are monitoring a system, and we are given a utility/cost function for comparing predictions made about this system to what happens really. For example, we are monitoring the weather. We have a pay-off function  $U(y, z)$  related to setting an equipment outdoor, with  $y$  the weather we predict to take our decision and  $z$  what really happens. We benefit from the equipment in the case it is outside when the weather is good, so  $U(y = \text{sunny}, z = \text{sunny}) = 1$ , while we gain nothing when it is inside and it is raining:  $U(y = \text{rain}, z = \text{rain}) = 0$ . We miss an opportunity when we keep the equipment indoor when it could have been useful, so  $U(y = \text{rain}, z = \text{sunny}) = -1$ . The equipment gets damaged under the rain, so  $U(y = \text{sunny}, z = \text{rain}) = -2$ . We would like events telling us when to set up the equipment or not

based solely on the current system state  $x$ . These events are determined by maximising the expected utility of our predictions  $y$  based on  $x$ .

This simple scenario is easy to transpose to more elaborated contexts. This article presents the theoretical background for this problem, as well as a concrete algorithm for computing the above events from data and a utility function only. The main contributions of this article are :

- A practical algorithm for computing the system states from data, which as an intermediate step reconstructs the  $\epsilon$ -machine [12].
- The way information is encoded in the utility function, which represents a new and clear way to represent the user knowledge independently of the system's intrinsic dynamics.
- The decisional states concept, allowing a modeller to represent system states with equivalent decisions for the user based on the preceding utility function.

Section 2 relates this context to background information and existing concepts in the literature. Section 3 introduces formally how the knowledge brought by a utility function can be used in order to compute the internal states of the process leading to the same decisions. Section 4 gives mathematical examples of the theory introduced in Section 3. Section 5 details how to effectively compute decisional and other utility-related process states from data. Section 6 gives application examples, including inferring hidden states from symbolic time series, detecting patterns in cellular automata and edges in images. A general conclusion is then given.

Free/libre source code is available and a link to the reference implementation is given at the end of the document.

## 2 Background information

### 2.1 Causality, predictions and causal states in a physical context

Let us consider what a prediction means in a physical framework, where information transfer is limited in speed.

---

\*University of Rennes 1. Part of this work was done while working at the LTSI laboratory, part while working at the CAREN laboratory.

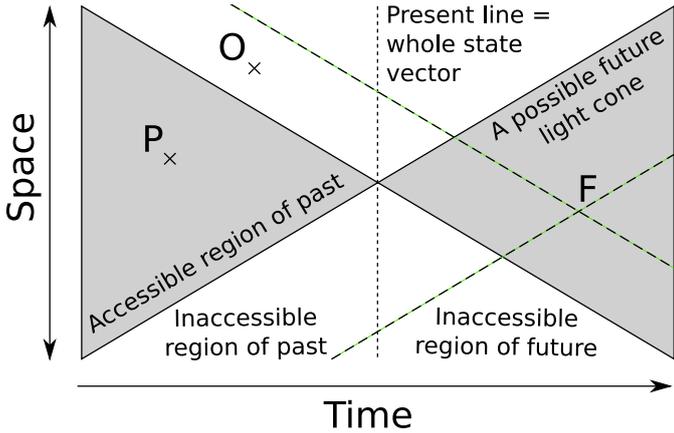


Figure 1: Light-cone representation of a prediction problem

Fig. 1 displays a schematic view of a system’s past and possible future.

In this view the system present is a single point in state-space. Contrast this with dynamical systems where the present is the whole state vector, the middle vertical line in Fig. 1. Here there is no instant propagation of information, and only a small portion of the state vector is accessible.

The past light-cone is the collection of all points that could possibly have an a priori causal influence on the present. The future cone is the collection of all points that might possibly be influenced by the present state. The problem is that in order to infer correctly the state of a point  $F$  in the future cone we might potentially need all points in the past light cone of  $F$ . It would theoretically be possible to have access to the points like  $P$  in the system current past, provided in practice that we indeed recorded the value of  $P$ . However there is by definition no way of getting the value of points like  $O$  that are outside the current system past light cone. Since both points belong to the past light cone of a point  $F$  in our own future, the consequence is that even for deterministic systems we get a statistical distribution of possible futures for a given observed past, depending on what information present outside the current past cone is necessary to predict the future. In other words, boundary and/or conditions in inaccessible regions may determine part of the future, which is well-known in physics.

Let us now consider grouping two past cones  $x_1^-$  and  $x_2^-$  together if they lead to the same distribution of futures  $P(X^+|x_1^-) = P(X^+|x_2^-)$ <sup>1</sup>. Suppose that point  $P$  in the past have a distinct value for pasts  $x_1^-$  and  $x_2^-$ . Then there is no way to recover what the value of  $P$  was by new observations: we cannot use future knowledge to decide

<sup>1</sup>In this document an upper-case  $P$  is used to denote the whole distribution, lower-case  $p$  is used to denote the probability density at one point. Similarly an upper-case  $X$  denotes a whole space, whereas  $x \in X$  denotes a point in that space.

between  $x_1^-$  and  $x_2^-$  since  $P(X^+|x_1^-) = P(X^+|x_2^-)$ . For all practical matter these two pasts are then equivalent. Mathematically the associated equivalence relation  $x_1^- \sim x_2^-$  partitions the system past light cones in sets  $\sigma(x^-) = \{y^- : P(X^+|y^-) = P(X^+|x^-)\}$ .

The sets  $\sigma$  are called the *causal states* of the system [24]. In a discrete scenario a new observation leads to a transition from a state  $\sigma_1$  to a state  $\sigma_2$ . The causal states and their transitions form a deterministic automaton: the  $\epsilon$ -machine [12]. A neat result is the abstraction of the time dependencies into the states. The transitions between states include all dependencies from the past that could have an influence on the future, hence the  $\epsilon$ -machine actually forms a Markovian automaton [24, 23].

However unlike a traditional Hidden Markov Model (HMM), the  $\epsilon$ -machine strongly constrains the state-to-state transitions by construction. The internal states of the  $\epsilon$ -machine thus have a very definite meaning (see above) and the structure of the  $\epsilon$ -machine can be inferred from data (see Section 5). The hidden states of the HMM are by contrast a free parameter when training the model to best fit the data, which needs to be estimated by techniques like maximum-likelihood or cross-validation [11]. The differences between finite state-output Markov processes and  $\epsilon$ -machines is expressly highlighted in [23]. The interested reader may find a comparison with alternative techniques like Variable-Length Markov Models [9], Observable Operator Models [16], Predictive State Representations [19], as well as extensions in [25, 21, 31, 20] for example.

The causal state construction based on light-cones was introduced in [26]. This framework is well suited to the analysis of physical contexts like fluid dynamics [17], and when causal relations can naturally be traced back like in a neural network context [6]. Yet the causal states construction is not limited to light-cones. We can also cluster together data points  $x \in X$  according to the conditional distributions  $P(Z|x)$  of points  $z \in Z$  in a space of predictions. The same equivalence relation as above can be defined, except that now care must be exercised on the interpretation: all we have defined are internal states with the same predictive power, without referring to causality. For example, sneezing and coughing are good predictors for being ill, though they are the symptoms and not the cause of the illness. However, when the space  $X$  is restricted to past time (and  $Z$  to future time) as is the case in this section, it is a reasonable assumption that a causal relation indeed provides the desired predictive ability. We refer to the equivalence classes induced by the above relation as causal states in this document, following the current usage of the term, but keep in mind that prediction and causation are different issues.

## 2.2 Decisions using a utility function

The idea of using the expected utility in order to determine which decision to take is not new<sup>2</sup>. Usually the utility is defined as a real-valued function associated to each outcome and quantifies the user’s interest in that outcome. Probability theory is then used to estimate the expected utility one may get when taking different actions. The action with maximal expected utility is then usually retained, although risk-aversion effects are sometimes taken into account. There is an abundant literature on probability and decision theory [2, 5, 32, 10, 15] which details these ideas.

There also exists numerous ways to combine a utility function with a Markov model. In a (possibly Partially Observable) Markov Decision Processes (MDP) [15, 10], the previous observations are used to build a model of the world (possibly with hidden states). Actions are then chosen according to this model in order to maximise the expected utility that would result of that choice (including costs and rewards). MDP consider a feasible set of actions that can be taken at any internal state, and the effect of these actions. The utility can be expressed as a *single-parameter function of the past* histories  $U(x^-)$  or it can be expressed in terms of a reward  $R(S = s)$  of being in state  $s$  and a cost function  $C(S = s, A = a)$  of taking action  $a$  while in state  $s$ . In [33] each hidden state of the Hidden Markov Model is attributed a mean and variance utility of the process to be in that state. This approach fares well when the utility is itself a global quantity and may evolve over time.

In the present framework, previous observations are also used to build a model (the  $\epsilon$ -machine). However, the  $\epsilon$ -machine captures equivalence classes of probability distributions of futures by definition. Hence, all past observations within the same state will lead to the same optimal decisions (see Section 3.2): these decisions are taken based on predictions of what will happen next. The utility  $U(y^+, x^+)$  is thus here a *two-parameter function of the future*, and quantifies the benefits/costs incurred by comparing what we thought would happen (a predicted future  $y^+$ ) to what really happens (the true future  $x^+$ ). An common assumption with the previous approaches is that the utility function encodes all the information needed to take a decision. In the present case, when both the expected utility and the predictions are the same we assume the user takes the same decisions.

When dealing with utility functions based on the effect of predictions the  $\epsilon$ -machine naturally becomes the underlying model that is inferred from data. In other words the utility function determines a structure corresponding to the user knowledge on top of the  $\epsilon$ -machine, while the  $\epsilon$ -machine itself represents the system’s internal relations

<sup>2</sup>The Wikipedia entry traces the history of the concept back to Bernoulli’s work in the 18th Century: [http://en.wikipedia.org/wiki/Expected\\_utility](http://en.wikipedia.org/wiki/Expected_utility)

independently of the user. This clear separation of internal structure vs. external knowledge is a neat secondary effect of defining the utility in terms of the effect of user predictions instead of attributing a utility directly to each outcome.

The decisional states framework is well suited to the scenario given in introduction, but perhaps not so well suited to reinforcement learning [22, 30]. However the same formalism is applicable to any system in which a prediction “usefulness” can be defined, including classical loss/utility functions like the minimum sum of squares error between the prediction and the actual future (See section 4).

A framework that is closely related to reinforcement learning and that also makes use of  $\epsilon$ -machines has recently been proposed [29]. It shows that a balance between exploration and exploitation emerges as a consequence of using the  $\epsilon$ -machine formalism without having to introduce that balance explicitly. That framework also relates learning with energy minimisation, and it makes explicit the agent actions. The present approach differs in that it introduces utility functions and considers that all futures are not equivalent for the agent. It would be interesting to try combining both approaches.

## 2.3 Practical implementation

A major contribution of this paper is to present a new algorithm for reconstructing  $\epsilon$ -machines and their extension to the decisional states introduced in this document (See Section 5). This REMAPF (Reconstruction of Epsilon MACHines in a Predictive Framework) algorithm offers more flexibility in its data representation and choice of parameters than the previous one CSSR (Causal State Splitting Reconstruction) [27], while providing a computational performance that makes it suitable for a large class of practical applications (See the examples in Section 6). It is possible to call only the  $\epsilon$ -machine reconstruction part (See Section 6.1) of the algorithm, and thus apply it to other frameworks than the one presently considered.

## 3 Decisional states framework

The previous section aimed at giving the reader an intuitive idea of what the framework proposed in this document is about. This section describes the framework formally.

### 3.1 General problem targeted by the proposed framework

Let  $X$  be a space comprising configurations  $x$  of the system under investigation. Let  $Z$  be a space of all entities that we wish to predict from the current system state. For example:

- In a symbolic series context,  $X$  is the set of all past strings up to the current point ( $x \in \mathcal{A}^*$  with  $\mathcal{A}$  the alphabet), and  $Z$  is the set of all future strings after the current point. The concrete example in Section 6.1 highlights this case.
- More generally for temporal systems with state space, an  $x \in X$  should include all causal influence from the past that might possibly affect the present (i.e, a past light cone, see Section 2.1). Similarly,  $z \in Z$  is the set of future light cones. The concrete example in Section 6.2 highlights this case.
- In the case of a non-temporal system,  $X$  is defined as the relevant space of parameters that have an influence on the system state at the point under investigation, and similarly for  $Z$  being the space of parameters influenced by  $X$ . A common example in physics is the Markov Random Field representation of lattice systems [4]. In an image context  $X$  is the neighbourhood of a given pixel up to a range that we assume determines the statistical distribution of that pixel, and  $Z$  is the value of the pixel [1]. The concrete example in Section 6.3 highlights this case.

For each configuration  $x \in X$ , we'd like to associate a prediction  $y_x \in Z$  amongst all possible outcomes. The actual outcome  $z \in Z$  can differ from  $y_x$ : we have a range of possible  $z \in Z$ , and they occur with a probability distribution  $P(Z|x)$ . This assumption restricts the spaces  $X$  and  $Z$  we may consider, for example discrete spaces, or continuous spaces with a canonical reference measure.

Let us now consider that the utility incurred by having acted according to prediction  $y$  when  $z$  is the future that actually happens is quantified by  $U(y, z)$ , independently of the particular  $x$  for which  $y$  was chosen instead of  $z$  (so,  $U$  is a real-valued function defined on  $Z^2$ ). We could equivalently define a loss function with  $L(y, z) = -U(y, z)$ . Minimising the loss is equivalent to maximising the utility, both concepts will be used interchangeably when needed.

An important difference between the present context and typical decision-theory frameworks (ex: [15, 33]) is thus that utility functions have two arguments: *the utility quantifies our knowledge of how bad it is to make mistakes*. Actions are based on predictions on what we think will happen, and are thus mapped to subsets of possible futures (ex: “going out for a hike” is mapped to “it won't rain in the next hours”). Actions are implicit in the utility function: The utility function quantifies the effect of having taken an action based on a prediction  $y$ , while  $z$  actually happens.

We can recover a scalar quantity at the current system state by computing the expected utility, integrated over all possible futures that may happen. The expected utility, in a continuous context where the integrals exist (ex: Lebesgue measurable spaces), is:

$$\mathbb{E}[U] = \int_{x \in X} \int_{z \in Z} U(y_x, z) p(x, z) dx dz$$

or in a discrete scenario:

$$\mathbb{E}[U] = \sum_{x \in X} \sum_{z \in Z} U(y_x, z) p(x, z)$$

And for all  $x \in X$  with non-null probability<sup>3</sup>:

$$\mathbb{E}[U] = \sum_{x \in X} p(x) \sum_{z \in Z} U(y_x, z) p(z|x) \quad (1)$$

The goal is usually to find a function  $y_x$  that maximises the expected utility: this would correspond to making the best predictions on average (and implicitly acting accordingly). By analogy with causal states [12] and  $\epsilon$ -machines, we now cluster together configurations  $x$  according to their statistical properties and look at conditions for which these clustering lead to maximal expected utility  $\mathbb{E}[U]$ .

### 3.2 Equivalence relations

$\mathbb{E}[U]$  is maximal when each term  $T(x, y) = p(x) \sum_{z \in Z} U(y, z) p(z|x)$  is maximal (see Eq. 1). Since  $p(x)$  is constant for a given  $T(x, y)$ , and assuming we can choose the  $y$  for each  $x$  independently, maximising  $T$  is equivalent to maximising each  $\sum_{z \in Z} U(y, z) p(z|x)$ . Let us note  $\mathbb{U}(y|x) = \mathbb{E}_{z \in Z}[U(y, z)|x] = \sum_{z \in Z} U(y, z) p(z|x)$ , the expected utility of choosing the prediction  $y$  for a given  $x$ .

Another assumption is implicit in this argumentation: that making a decision does not modify the system. The weather forecasting example in the introduction falls in this category. However, sometimes taking a decision modifies the system. For example, when monitoring a patient's health in order to decide whether to administrate a drug or not. In that case we have to rely on approximations (usually an additional assumption that the change is effective only at a different time scale than that of the observations) so we can still aggregate them on a recent past sliding window. Alternatively, other frameworks like Interactive Learning [29] might be better suited for these situations.

Let us now recall the causal states construction [24]:

**Causal state equivalence relation:**  $x_1 \stackrel{c}{\sim} x_2$  if, and only if, the conditional distributions  $P(Z|x_1) = P(Z|x_2)$  are the same. The equivalence classes  $\sigma(x) = \{w : P(Z|w) = P(Z|x)\}$  are called the *causal states*. See Section 2.1 for a discussion on this term.

By analogy with the causal states construction, let us now define the following equivalence relations:

**Utility equivalence relation:**  $x_1 \stackrel{u}{\sim} x_2$  if, and only if,  $\max_{y \in Z} \mathbb{U}(y|x_1) = \max_{y \in Z} \mathbb{U}(y|x_2)$ . That is, the

<sup>3</sup>Technically we should introduce here a set  $X' = X \setminus \{x : p(x) = 0\}$  of all  $x \in X$  with non-null probabilities. In practice we are dealing with observed system configurations with non-null probabilities, and will act as if  $X' = X$ .

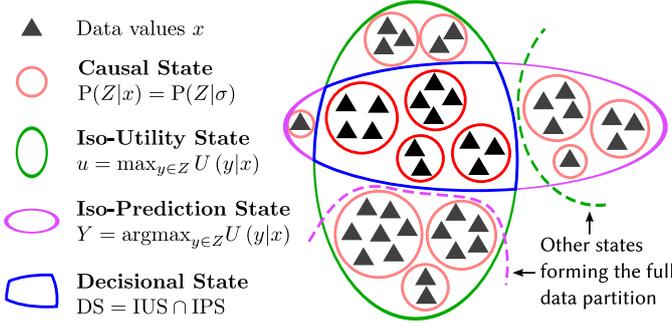


Figure 2: Relations between the different states.

maximal expected utility is the same at points  $x_1$  and  $x_2$ , even if the sets of optimal predictions  $Y(x_1) = \operatorname{argmax}_{y \in Z} U(y|x_1)$  and  $Y(x_2)$  that induce this utility might differ for  $x_1$  and  $x_2$ .

**Prediction equivalence relation:**  $x_1 \stackrel{p}{\equiv} x_2$  if, and only if,  $\operatorname{argmax}_{y \in Z} U(y|x_1) = \operatorname{argmax}_{y \in Z} U(y|x_2)$ . That is, the sets of optimal predictions  $Y(x_1) = Y(x_2)$  are the same, even if the utility induced by these predictions might differ for  $x_1$  and  $x_2$ .

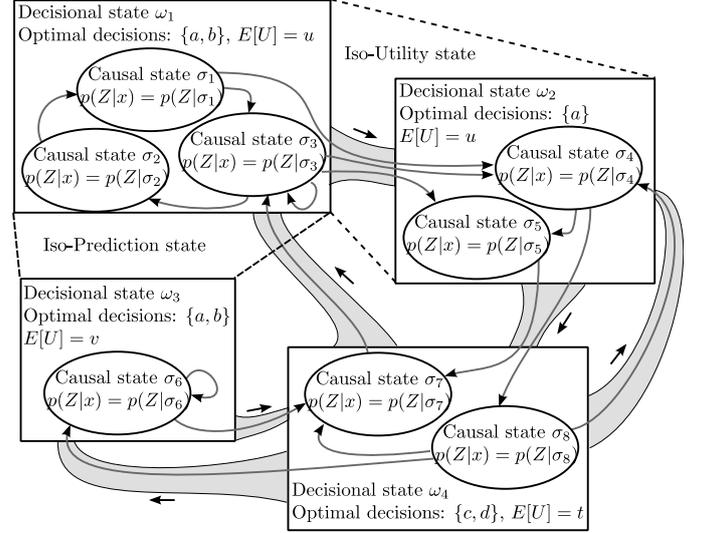
Let us call *iso-utility states*  $v \in \Upsilon$  and *iso-prediction states*  $\psi \in \Psi$ , the partitions of  $X$  corresponding to these equivalence relations:  $v(x) = \{x' : x' \stackrel{u}{\equiv} x\}$  and  $\psi(x) = \{x' : x' \stackrel{p}{\equiv} x\}$ .

Let us call *decisional states*  $\omega \in \Omega$  the intersection of both:  $\omega(x) = \{x' : x' \stackrel{p}{\equiv} x \text{ and } x' \stackrel{u}{\equiv} x\}$ . When both the expected utility and the optimal predictions are the same, we assume the decisions that are taken on the system are the same, hence the name. In other words, we suppose the utility function encodes all that a user needs to take a decision.

These equivalence relations partition the configuration space  $X$  into clusters, with the corresponding properties common to all points in the cluster. It should be noted that  $\mathbb{E}[U]$  as defined on the whole space does not consider which specific decisional state the process is in. Knowing which is the current cluster for any given point  $x$  allows us to refine the expected utility to a local  $\mathbb{E}_v[U]$  (with  $x \in v$  the iso-utility state) and which decision to take to reach this utility (by refining again to the decisional state). Section 3.4 details how to derive notions of complexity from these local expected values.

### 3.3 Relation between the causal, iso-utility, iso-prediction and decisional states

Let  $x_1$  and  $x_2$  be in the same causal state  $\sigma$ . Then by definition of the causal states  $P(Z|x_1) = P(Z|x_2)$ . In that case, the expected utility of any prediction  $y \in Z$  is the

Figure 3: Decisional states transition graph on top of the  $\epsilon$ -machine.

same for  $x_1$  and  $x_2$ :  $U(y|x_1) = \sum_{z \in Z} U(y, z) p(z|x_1) = \sum_{z \in Z} U(y, z) p(z|x_2) = U(y|x_2)$ . Therefore the optimal predictions and induced utilities are the same:  $x_1 \stackrel{p}{\equiv} x_2$  and  $x_1 \stackrel{u}{\equiv} x_2$ , and so is the combination of both.

Thus the causal states sub-partition both the iso-utility, iso-prediction and decisional states.

The converse is not true: we can have two distinct causal states  $\sigma_1$  and  $\sigma_2$  with the same maximum value of  $U(y|x) = \sum_{z \in Z} U(y, z) p(z|x)$  at the same  $y$  points, but with different  $p(z|x)$  for at least one  $z \in Z$ .

Figure 2 shows the relations between the different states defined on the process.

### 3.4 Transition graphs

In the discrete case the causal states form a deterministic automaton, the  $\epsilon$ -machine [12]. Since the iso-utility, iso-prediction, and decisional states are coarser partitions than the causal states each of them comprises one or more nodes of the  $\epsilon$ -machine automaton. The transitions between these nodes are either internal to the coarser state, in which case they are ignored (or represented as self-loops), or lead to another coarser state.

But the  $\epsilon$ -machine is a deterministic automaton: for each initial causal state  $\sigma_1$  and each discrete symbol  $a \in \mathcal{A}$ , in a discrete scenario with alphabet  $\mathcal{A}$ , there is at most one transition from  $\sigma_1$  labeled by  $a$  and it leads to a unique causal state  $\sigma_2$ , possibly the same as  $\sigma_1$ . If  $\sigma_1$  and  $\sigma_2$  belong to different coarser states, the transition is also present at the coarser level. However there may be several transitions with different labels between the same coarser states: in Fig. 3 this is the case for the transitions  $\sigma_4 \rightarrow \sigma_8$  and  $\sigma_5 \rightarrow \sigma_7$ . These transitions are not distinguishable at the coarser level: both are included in  $\omega_2 \rightarrow \omega_4$ . We

therefore loose the symbol labeling of the transitions at the coarser level. Figure 3 shows decisional states as a coarser level of causal states of an underlying  $\epsilon$ -machine, with iso-utility and iso-prediction states on top of the decisional states.

Let  $x$  be the current system configuration, and  $\omega$  the current system decisional state (so  $x \in \omega$ ). Assume symbol  $a \in \mathcal{A}$  is observed at this point. Then  $xa = s \in X$  is the system configuration after the observation. Let  $\gamma$  be the decisional state for  $s$ . Then  $p(e_{\omega \rightarrow \gamma}) = \sum_{x \in \omega} \sum_{a \in \mathcal{A}} p(xa \in \gamma | x \in \omega)$  is the probability of the transition event  $e_{\omega \rightarrow \gamma}$  from state  $\omega$  to state  $\gamma$ . The same construction also works for iso-utility and iso-prediction states:

- Iso-utility state transitions are events that change the expected utility, irrespectively of the implied symbols. Several causal states might belong to the same iso-utility state, as depicted in Fig. 2.
- Iso-prediction state transitions are events that change the possible optimal prediction choice, with the same comment.
- Decisional state transitions change at least one of the above.

An important consequence of this definition is that we loose the Markov property of the coarser graph<sup>4</sup>. For example, in the Fig 3,  $p(\Omega_t = \omega_1 | \Omega_{t-1} = \omega_4) \neq p(\Omega_t = \omega_1 | \Omega_{t-1} = \omega_4, \Omega_{t-2} = \omega_3)$  as in this later case (coming from  $\omega_3$ ) the internal causal state of  $\Omega_{t-1} = \omega_4$  is necessarily  $\sigma_7$ . Nevertheless, the coarser states are still defined as the minimal supersets of causal states leading to the same decisions. Knowing the epsilon machine in addition to the superstructure may lead to improved predictions of the future transitions, but it does not change the utility and optimal values associated to the current state.

In computational mechanics [12] the mutual information  $C = I(x; \sigma)$  between a configuration  $x$  and the causal state  $\sigma$  for  $x$  is referred to as the statistical complexity. In the discrete case, by definition  $C = I(x; \sigma) = H(\sigma) - H(\sigma|x)$  with  $H$  proper entropies (differential entropies in the continuous case). But then  $H(\sigma|x) = 0$  by construction of the  $\sigma$ , which in the discrete case leads to well-defined transition graphs. Thus  $C = H(\sigma)$  in this case, the amount of information necessary to encode the epsilon machine.

In the present context we might define by analogy a *decisional complexity*  $D$  as the amount of information necessary to encode the coarser level graph, *given a utility function*. Once that utility function is fixed we can compute the decisional states and define  $D = I(x; \omega)$  where  $x$  is a configuration of the system and  $\omega$  the decisional state for  $x$ .

$$\begin{aligned} \text{Since} \quad & -p(\omega) \log_2 p(\omega) & = \\ - \left( \sum_{\sigma \in \omega} p(\sigma) \right) \log_2 \left( \sum_{\sigma \in \omega} p(\sigma) \right) & \leq \end{aligned}$$

<sup>4</sup>Thanks to one of the anonymous reviewers for pointing this example and highlighting this aspect of the coarser graphs.

$-\sum_{\sigma \in \omega} p(\sigma) \log_2 p(\sigma)$  because  $-\log_2$  is monotonically decreasing, then  $D \leq C$  in the discrete case.

The same construction also works for defining similar quantities:

- $P = I(x; \psi)$  is the tentatively called here the optimal prediction complexity.
- $V = I(x; v)$  is the complexity of estimating the expected utility from  $x$ .

Similarly to [26] it is possible to define a local statistical complexity measure corresponding to each state:  $C_\sigma = -\log_2(p(\sigma))$  in bits, and correspondingly for the other coarser states:  $D_\omega = -\log_2(p(\omega))$ , similarly for  $P_\psi$  and  $V_\nu$ . With this definition the global values are simply the expectation of the local values over all states (ex:  $D = \sum_{\omega} p(\omega) D_\omega$ ). The local complexity measures are used for the examples in Section 6.

### 3.5 Interpretation and notes

Decisional states are equivalent to merging those causal states which lead to the same decisions relatively to our utility function. In this case the causal states have lost their maximality property due to the fact we're only interested in making a prediction and not in keeping the full conditional distributions. We have, in the general case, clustered together the causal states that lead to the same optimal predictions and maximal expected utility value, based on a given utility function.

Conversely this defines an equivalence relation amongst utility functions: Two utility functions  $U_1$  and  $U_2$  are equivalent when they induce the same clustering of causal states into the decisional ones, with the same expected values and optimal predictions. These utility functions would induce the same decisions in a system: they are functionally equivalent. Isomorphisms between utility functions leading to the same predictions but with different utility values could also be defined: these are transformations of utility functions that preserve the iso-prediction states. Similarly, transformations could be defined that only preserve the iso-utility states.

The transitions between the iso-utility states correspond to events that provoke a change in the expected utility of the system. Identifying these events might become a crucial practical application, for example for detecting when the expected utility reaches a predefined threshold.

The transitions between the iso-prediction states correspond to events that provoke a change in the optimal predictions that can be chosen. Similarly, a user might be interested in monitoring these changes, for example, to maintain the current action as long as it is appropriate (as long as it matches one of the possible predictions for the system's evolving iso-prediction state).

The hypothesis made here is that when the cost/utility is defined in terms of a functional (high-level) value, when

it has a signification in high-level terms, then the transition events also correspond to interesting high-level objects to look at. Ignoring the internal transitions and keeping only the coarser-level transitions also leads to a time-scale change in practice. This might form the basis for an automated search for meaningful events in a given system's evolution directly at a higher-level than that of its constituents.

In any case, the utility function encodes external information not available in the original data. So long as one stays with causal states, only information present in the low-level data can be obtained. Much like introducing a prior in a Bayesian framework, here the utility function can be seen as encoding an a priori information not available in the original data. This has at least three consequences from an emergentist point of view:

- The causal states represent the finest scale at which we can meaningfully associate a utility function and take decisions. Since the decisional states are supersets of the causal states, then any partition of the data defined with respect to a utility function cannot go below that scale, whatever the chosen utility function.
- Macro-level information needs not be computationally reducible [7] to the lower level in order to be incorporated: The utility function is defined on  $Z^2$ , not  $X$ , and it can possibly be incompressible, stated as a value table and not explicitly computed in terms of the lower-level scale. The data  $x \in X$  is then clustered into sets which need not have a meaning defined at that level.
- If the hypothesis that “emergent structures are sub-machines of the  $\epsilon$ -machine” [24, sec. 11.2.2] is correct, then the decisional states are the emergent structures corresponding to a given utility function. Rather than looking for emergent entities directly we might then encode our knowledge in a utility function, and look at the decisional states in order to find good emergent entity candidates. If these do not suffice, we might then refine the utility function iteratively.

Finally, it should be noted the utility function is not the only source of external introduction of knowledge in the system. Additional assumptions are made either implicitly or explicitly if the system is able to generalise to unknown values. For example, the hypothesis that  $P(Z|x)$  can be decomposed using kernels or Bayesian networks could be one such assumption. The accuracy of the proposed method for finding decisional states depends on how well these extra assumptions are verified, independently of the chosen utility function.

The algorithm proposed in Section 5 does not handle the verification of preconditions, which are expected to be performed by the user depending on the context (ex: nature of the data). However the reference implementation

(link given in Appendix) is fully generic and allows testing different sampling and generalisation methods if needed.

## 4 Analytic examples

### 4.1 Example 1: when bad predictions are useless

In this subsection utility is given to a prediction only if it is correct; otherwise, the prediction is declared useless:  $U(z, z) = 1$  and  $U(y, z \neq y) = 0$ . In a continuous scenario the delta function  $U(y, z) = \delta(y, z)$  is used instead.

From Section 3.2:

$$U(y|x) = \sum_{z \in Z} U(y, z) p(z|x)$$

then becomes:

$$U(y|x) = p(y|x)$$

The set  $Y(x)$  of predictions  $y$  realising an optimal gain becomes:

$$Y(x) = \{y : p(y|x) = \max_{z \in Z} p(z|x)\}$$

And Eq. (1) leads to:

$$\mathbb{E}_{max}[U] = \sum_{x \in X} p(x) \max_{z \in Z} p(z|x)$$

But for each causal state  $\sigma \subset \omega$  in each decisional state  $\omega$  the conditional probability  $P(Z|x) = P(Z|\sigma)$  is the same for all  $x \in \sigma$ . The decisional states are found by gathering causal states with the same maxima points  $y$  for  $P(Z|\omega)$ . We can then write in this special case the above formula as:

$$\mathbb{E}_{max}[U] = \sum_{\omega} \sum_{\sigma \subset \omega} p(\sigma) p(y_{\omega}|\sigma)$$

where  $y_{\omega}$  is taken as any maxima of  $p(Z|\sigma)$  common to all  $\sigma \subset \omega$ .

Under the condition  $U(z, z) = 1$  and  $U(y, z \neq y) = 0$  (or  $U(y, z) = \delta(y, z)$  in the continuous case) the full conditional probability distributions  $p(Z|x)$  do not matter, what's important is that these distributions peak at the same maxima.

### 4.2 Example 2: Loss defined by error squared

This section investigates the case where the loss function  $L(y, z)$  can be written as a squared difference between the actual event and the prediction:  $L(y, z) = (z - y)^2$ , provided this operation is meaningful in  $Z$ .

We re-develop the treatment from [5, section 1.5.5] in our new context:

With the above loss function Eq. 1 becomes:

$$\mathbb{E}[L] = \int_{z \in Z} \int_{x \in X} L(y_x, z) p(x, z) dx dz$$

$$\mathbb{E}[L] = \int_{z \in Z} \int_{x \in X} (z - y_x)^2 p(x, z) dx dz$$

As in Section 3.1, the goal is to find an  $y_x$  function that minimises the expected loss:  $\mathbb{E}_{min}[L]$ .

The extrema of  $\mathbb{E}[L]$  are given by the functional equation  $\frac{\partial \mathbb{E}[L]}{\partial y_x} = 0$ , with:

$$\frac{\partial \mathbb{E}[L]}{\partial y_x} = 2 \int_{z \in Z} (z - y_x) p(x, z) dz$$

Solving  $\frac{\partial \mathbb{E}[L]}{\partial y_x} = 0$  gives:

$$\int_{z \in Z} z p(x, z) dz = y_x \int_{z \in Z} p(x, z) dz$$

So except for a set of  $x$  with null probability mass:

$$p(x) \int_{z \in Z} z p(z|x) dz = y_x p(x)$$

$$y_x = \mathbb{E}_{z \in Z}[z|x] \quad (2)$$

For a given causal state  $\sigma$ ,  $P(Z|x)$  is the same for all  $x \in \sigma$  so we can write  $y_\sigma = \mathbb{E}_{z \in Z}[z|\sigma]$ .

The decisional states are in this example obtained by clustering together the causal states with the same expected value of  $z$  within the state.

These results are obtained because the utility function can be treated analytically; in the general case we do not have such simple formula available. The next section presents an algorithm that can infer the structure of the decisional states from observed data and numerical integration.

## 5 Estimating the decisional states from data

### 5.1 General presentation of the algorithm

There are two main critical tasks the algorithm must perform:

- Estimating the probability distributions  $P(Z|x)$  from data. The probability distribution estimator is responsible for providing values for unobserved data (generalisation ability). It might use all available observations:  $\hat{P}(Z|x) = F(\mathcal{O})$ , where  $\mathcal{O} = \{(\mathbf{x}_i, \mathbf{z}_i)_{i=1 \dots N}\}$  represents the data in the form of observation pairs  $(\mathbf{x}_i, \mathbf{z}_i)$ , and  $F$  a generic function.
- Clustering  $X$  into causal, iso-prediction, iso-utility and decisional states according to the user needs. This implies as a sub-task estimating the maxima for  $\mathbb{U}(y|x)$ . The first step is to build  $\hat{\mathbb{U}}(y|x) =$

$\int_Z \hat{p}(z|x) U(y, z) dz$  with a user-provided integrator and the utility function. Then, an optimiser might be invoked so as to compute  $Y(x) = \text{Argmax}_{y \in Z} \hat{\mathbb{U}}(y|x)$ .

So, in summary, the user must provide (explicitly or implicitly using the reference implementation default choices):

- A probability density estimator  $\hat{P}(Z|X)$  from data observations  $\mathcal{O}$ .
- A utility function  $U$  acting on  $Z^2$  with  $Z$  the space of predictions.
- An integrator for computing the expected value of  $U$  with respect to the estimated density.
- A multi-modal optimiser in order to compute  $Y(x) = \text{Argmax}_{y \in Z} \hat{\mathbb{U}}(y|x)$ .
- A clustering algorithm for gathering probability distributions (for causal states), utility values (for iso-utility states), or similar sets  $Y(x)$  (for iso-prediction states). Decisional states are found by intersection of the iso-utility and iso-prediction states.
- Optionally, the user may associate a symbol to each transition  $(\mathbf{x}_t, \mathbf{x}_{t+1})$ . This step is detailed in Section 5.5, once the main algorithm is explained and we can see why and when this step may become necessary.

Figure 4 recapitulates these points and shows the algorithm steps that will be detailed in the next sections. Readers not used to generic programming might be suprised by the functional inputs: The reference implementation proposes default choices for these functions, but the user is free to provide any equivalent replacement code if so desired. The algorithm in Fig. 4 is described in general terms but it is well-defined and a concrete reference implementation in C++ is provided, see the Appendix.

### 5.2 Kernel density estimation

This section describes one way to perform Step 1 in Fig. 4.

A discrete probability estimator is suitable for small  $X$  spaces where a sufficient amount of data was observed, so that  $P(Z|X)$  can be reliably estimated by counting occurrences of all  $x$  and  $z$ . For larger spaces or when unknown or continuous  $X$  might be encountered, the system must be able to generalise. We now present the case for a Kernel Density Estimation (KDE) [28] of the probability density  $\hat{P}(Z|X) = F(\mathcal{O})$ .

In general, the kernel  $K(a; b)$  with  $a$  and  $b$  in the joint space  $\{a, b\} \subset X \times Z$  is not separable: The density estimate is  $\hat{p}(x, z) \propto \sum_{i=1}^N K(\mathbf{x}_i, \mathbf{z}_i; x, z)$ , summing over all observation pairs  $(\mathbf{x}_i, \mathbf{z}_i) \in \mathcal{O}$ . In the particular case of separable kernels for the configuration space  $X$  and the prediction space  $Z$  we have instead:  $\hat{p}(x, z) \propto$

Data inputs:

- Pairs of observations  $\mathcal{O} = \{(\mathbf{x}_i, \mathbf{z}_i)_{i=1 \dots N}\}$ ;
- (Optional): Symbols  $(\mathbf{s}_i)_{i=1 \dots N-1}$  for each transition  $(\mathbf{x}_i, \mathbf{x}_{i+1})_{i=1 \dots N-1}$ ;
- Parameters for the functional inputs (ex: threshold for matching probability distributions).

Functional Inputs:

- A probability density estimator  $PDE$  such that  $\hat{P}(Z|X) = PDE(\mathcal{O})$ . The distribution type is user-defined;
- A utility function  $U : Z^2 \mapsto \mathbb{R}$ ;
- An integrator  $Integ$  over  $Z$ ;
- A multi-modal optimiser  $Argmax$  over  $Z$ ;
- A clustering algorithm  $C1$  acting on probability densities  $\hat{P}(Z|X)$ ;
- A clustering algorithm  $C2$  over subsets of  $Z$ ;
- A clustering algorithm  $C3$  over  $\mathbb{R}$ .

Algorithm:

1. Build the density estimates  $\hat{P}(Z|\mathbf{x}_i)$  for each  $\mathbf{x}_i$  in the data set using  $PDE$ .
2. Cluster the density estimates using  $C1$  into causal states  $\hat{\sigma}$ .
3. (Optional) Refine the estimates  $\hat{\sigma}$  and loop to step 2 using the symbols  $(\mathbf{s}_i)_{i=1 \dots N-1}$ . See Section 5.5.
4. Average out  $\hat{P}(Z|\hat{\sigma}) = avg_{\mathbf{x}_i \in \hat{\sigma}} \hat{P}(Z|\mathbf{x}_i)$ . See Section 5.4.
5. Compute  $Y(\hat{\sigma}) = Argmax_y Integ_z (U(y, z) \hat{P}(z|\hat{\sigma}))$  for each causal state estimate  $\hat{\sigma}$ , retaining the utility  $U(\hat{\sigma})$  obtained for these maxima.
6. Cluster the causal states estimates using  $Y(\hat{\sigma})$  and  $C2$  into iso-prediction estimates  $\hat{\psi}(\hat{\sigma}) \in \hat{\Psi}$ .
7. Cluster the causal states estimates using  $U(\hat{\sigma})$  and  $C3$  into iso-utility estimates  $\hat{v}(\hat{\sigma}) \in \hat{Y}$ .
8. Intersect  $\hat{\Psi} \cap \hat{Y}$  into decisional states  $\hat{\Omega}$  that partition  $X$ .
9. (Optional) Produce the transition graphs, and the  $\epsilon$ -machine if the symbols are available.
10. (Optional) Compute the global complexities of the system  $C$ ,  $D$ ,  $P$ , and  $V$  from section 3.4.
11. (Optional) For each  $\mathbf{x}_i$ , compute the local complexity equivalents of  $C$ ,  $D$ ,  $P$ , and  $V$  at this point (see Section 3.4).

Figure 4: Decisional state reconstruction algorithm.

$\sum_{i=1}^N K^x(\mathbf{x}_i; x) K^z(\mathbf{z}_i; z)$ . Even when the kernel is separable the user may benefit from the joint kernel approach: For analysing time series it is natural to consider a moving window of  $dim(X \times Z)$  values and perform the density estimation on the joint space. In another example in Section 6.3 an image is considered as the limit distribution of a Markov Random Field [1], and the density estimation is also performed on the joint space (with  $Z$  being in that example the space of pixel values and  $X$  the space of pixel neighbourhoods).

In any case, the conditional probability density is estimated by integrating out the  $\hat{p}(x)$  factor over  $Z$ :  $\hat{p}(z|x) = \hat{p}(x, z) / \int_{\zeta \in Z} \hat{p}(x, \zeta)$ . Several sampling mechanisms are provided over  $Z$  for the integration, including exhaustive listing of  $Z$  for small search spaces. The adequate method depends on the particular user application.

Computing the causal states (and the other states built on top of the causal states) only requires the conditional distributions, and not the joint ones. Without loss of generality it is thus possible to request that  $K(a, a) = 1$  with  $a$  being  $x$  or  $z$  or the joint data  $(x, z)$  depending on the above cases. For example, the radial basis function  $K(a, b) = e^{-\|a-b\|^2/h}$ , with  $h$  the kernel width. Indeed, dividing by  $\hat{p}(x)$  absorbs the change of scale. Better numerical accuracy is however achieved by requesting  $K(a, a) = 1$ , especially in high dimensions where the multivariate Normal kernel would lead to very small  $K(a, a)$ .

The discrete case is recovered when choosing the delta function as a kernel. In that case, similar observations  $(\mathbf{x}_i, \mathbf{z}_i)$  are effectively summed up for a given  $\mathbf{x}_i$  and the probability estimator is an histogram. In practice it is preferable to use a specialised discrete estimator implementation for efficiency reasons.

Finally, the kernel width  $h$  can be chosen according to a variety of estimators from the data [13]. In practice it has been observed that results ultimately depends on the final task for which the algorithm is applied to.  $h$  is then considered as a free parameter, which can be determined for example by cross-validation or by using a genetic algorithm. This gave the best results for classification tasks based on the decisional complexity feature (ex: classification of EEG time series [8]). An hypothesis is that while the kernel width  $h$  found this way does not realise an a priori form of optima (like the AMISE [13]), it realises an a posteriori ideal compromise between bias and variance in the estimated density for the particular task the algorithm is applied to. This is similar to the approach in [3] except that we have reduced the meta-parameter search to  $h$  and got rid of the histogram boundaries by using a KDE.

The default implementation proposes a reasonable choice based on the average distance between nearest data points, from which the aforementioned cross-validation and search techniques can build on.

### 5.3 Using the probability estimates

Two operations are performed using  $\hat{P}(Z|x)$ :

- Comparison: We need to check whether  $\hat{P}(Z|x_1)$  and  $\hat{P}(Z|x_2)$  are similar for clustering or not  $x_1$  and  $x_2$  in the same causal state.
- Expectation: We need to estimate the expected utility of a prediction  $y \in Z$  for a given  $x \in X$ :  $\hat{U}(y|x) = \int_{z \in Z} \hat{p}(z|x)U(y, z)$ .

Comparison is handled by choosing a similarity measure between probability distributions. The reference implementation proposes the  $\chi^2$  statistic, the Bhattacharyya, Variational and Harmonic mean distances, and the Jensen-Shannon divergence [18]. The Bhattacharyya distance is the default for the Kernel Density Estimation, and a  $\chi^2$  test is the default for the discrete case.

Let  $S \subseteq Z$  be a set of sample points used over  $Z$  for comparing the probability distributions (possibly with  $S = Z$  for an exhaustive approach). Expectation of the utility for a candidate  $y \in Z$  is simply performed numerically over  $Z$  at the chosen sample points  $S \subseteq Z$ :  $\hat{U}(y|x) = (\sum_{s \in S} \hat{p}(s|x)U(y, s)) / (\sum_{s \in S} \hat{p}(s|x))$ .

### 5.4 Clustering

Clustering of the causal states is performed directly by matching the probability distributions as defined in the previous subsection.

For iso-utility and iso-prediction states an additional step is necessary: Optimising  $\hat{U}(y|x)$  in order to find  $Y(x) = \text{Argmax}_{y \in Z} \hat{U}(y|x)$ . Any multi-modal optimisation scheme can be invoked at this point. Equivalent best predictions  $y \in Y(x)$  must be found, so uni-modal search schemes returning only one candidate are not adapted. Once the prediction sets  $Y(x)$  and the optimal utility values are computed it is possible to cluster them.

False positives are when  $x_1$  and  $x_2$  are clustered together when they are mathematically not equivalent, false negatives are when the points are in different states when they should not. These risks are minimised by providing more sample points to look at and by increasing the data size. In the limit of an infinite number of data points and samples, consistency is determined by the chosen approximate matchers (ex: the Bhattacharyya distance in the previous section) and by whether the data respects or not the mathematical assumptions needed for the theory to work (ex: conditional stationarity of the  $P(Z|x)$ ).

Additionally we can exploit the fact that causal states sub-partition the decisional ones. If we compute the iso-utility, iso-prediction and decisional states first we might then restrict the search for similar probability distributions  $P(Z|x_1)$  and  $P(Z|x_2)$  to points  $\{x_1, x_2\} \subset \omega$  within each decisional state  $\omega$ .

If we compute the causal states first (as in Fig. 4), we might use representative  $\hat{P}(Z|\sigma)$  distributions for each

causal state, by averaging all  $\hat{P}(Z|x)$  for  $x \in \sigma$  in order to reduce numerical discrepancies:  $\hat{P}(Z|\sigma) = \text{avg}_{x \in \sigma} \hat{P}(Z|x)$ . The expected utility  $\hat{U}(y|x)$  is then set to  $\hat{U}(y|\sigma)$  for all  $x \in \sigma$ . This approach (causal states first) was found to give better results in practice.

The reference implementation provides two clustering algorithms. The first algorithm is simply the straightforward implementation of matching each candidate entity to each currently found cluster, and maintaining averages for the clusters as described above. Once each observation pair was processed the clusters themselves are matched with each other. This step reduces the risk of spurious clusters that appear as a side-effect of the implicit ordering in which the data is processed. The data may then be re-matched iteratively if so desired in  $P$  passes. This algorithm complexity is at worst  $O(P(KN + K(K-1)/2))$  with  $K$  the number of clusters and  $N$  the data size. It is thus *linear in data size*, and faster than the second algorithm proposed below. The inconvenient is a dependence on the data presentation order, although this can be minimised by randomising the data presentation order and by performing several passes as aforementioned. See also the discussion in the reference [26] where their implementation in Object Caml uses a similar but simpler clustering technique for computing causal states. The leading Causal State Splitting Reconstruction (CSSR) algorithm [27] in the domain also uses data order randomisation and argues for consistency in the limit of a large number of observations. See also [6] where the author proposes an incremental version of this first algorithm that is additionally able to handle data on sliding windows in order to cope with slowly non-conditional stationarity systems.

The second provided algorithm is a single-linked hierarchical clustering with a complexity in  $O(N(N-1)/2)$ . This is equivalent to finding connected components with respect to the given match predicate, similar to the DBSCAN [14] algorithm except that we must label each value to a cluster (DBSCAN leaves out some values as noise). Depending on the application one may prefer this algorithm to the first despite of its worse cost for the following reasons:

- Arbitrary cluster shapes, unlike the first algorithm where clusters are balls around the average value according to the similarity measure.
- The clusters are stable with respect to the data presentation order.
- Occam's razor: we want to find the simplest model able to handle the data. Connected components maximise the clusters size by gathering data when a matching path is found between them. This leads to a minimal number of states in the discrete case while ensuring that data in different states do not match (consistency), hence minimal statistical  $C = H(\sigma)$  or decisional  $D = H(\omega)$  complexity values. The down-

side is a sensitivity to the single-link effect: a single error in the data may join several clusters, hence increasing the number of the aforementioned false positives.

- An interpretation for the continuous case. Connected components ensure  $d(a, b) > \Delta$  for  $a$  and  $b$  points in different clusters,  $d$  a dissimilarity measure (ex: in utility values for iso-utility states, on probability distributions for causal states, etc), and  $\Delta$  a threshold for the mismatch between clusters. This is equivalent to single-linked hierarchical clustering where we cut the hierarchy at level  $\Delta$ . In the continuous case the transition graph construction fails on a continuum of infinitely many nearby states. In that case connected components with threshold  $\Delta$  ensure that the system state changed at least by that amount when transiting from one point to the next in a different component. For example when monitoring a system expected utility value, a decision might be taken only when a sudden change is detected, but not for a gradual change of the same magnitude.

Clustering is a research domain in itself and an important aspect of data mining. The reference implementation proposes the above two choices as their trade-off covers a vast range of usual cases. The user is welcome to plug in a custom algorithm: thanks to the generic nature of the reference implementation using C++ templates the clustering part is independent of the rest of the computations.

## 5.5 Ensuring the $\epsilon$ -machine determinism

Causal State Splitting Reconstruction (CSSR) [27] is the reference algorithm for reconstructing  $\epsilon$ -machines on discrete strings of symbols. It works by recursively splitting the current causal state estimates as the string length is increased. The consistency on shorter string lengths is maintained while the causal states are refined to take in account more symbols. In the limit it provably converges to the true causal states.

In the present case we do not act on strings of symbols but on  $(x, z)$  mappings. Hence it is not possible to refine iteratively the current causal state estimates by enlarging the dimensions of  $X$  and  $Z$ . Yet the “symbols” of discrete data are implicitly present in the  $(\mathbf{x}_i, \mathbf{x}_{i+1})$  transitions when monitoring the system (the index  $i$  corresponds for example to ordered time steps, but spatial transitions are possible as well). It would be possible to recover a symbolic representation of the data set from all such transitions<sup>5</sup>, and apply CSSR if so desired. Here we directly cluster the system configurations  $x \in X$ , not necessarily represented as strings of symbols. For example,

<sup>5</sup>For a discrete finite data set, assume  $x \in X$  is coded on  $N$  bits. The difference between  $\mathbf{x}_i$  and  $\mathbf{x}_{i+1}$  is always representable as a symbol in an alphabet with size at most  $2^N$ .

each  $x \in X$  might correspond to a past light cone (see Appendix A).

The drawback is that the proposed algorithm does *not* so far ensure that the resulting automaton is deterministic in terms of symbol transitions, which is a condition for being a valid  $\epsilon$ -machine [24, 21]. The labeled transitions between states can be recovered by looking at the symbol suffix implied by passing from  $\mathbf{x}_i$  to  $\mathbf{x}_{i+1}$ . But there is no guarantee at this point that a given (state+symbol) combination always lead to the same state deterministically.

Example: Suppose that  $\mathbf{x}_i = aaba$  and  $\mathbf{s}_i = abba$  are in the same causal state:  $P(Z|\mathbf{x}_i)$  and  $P(Z|\mathbf{s}_i)$  match and were clustered together with string length limited to depth 4. We observe that  $\mathbf{x}_{i+1} = abac$  and  $\mathbf{s}_{i+1} = bbac$ , with the same suffix  $c$ , and that  $P(Z|\mathbf{x}_{i+1})$  and  $P(Z|\mathbf{s}_{i+1})$  do not match anymore and therefore were not clustered in the same state. This is a violation of the  $\epsilon$ -machine determinism: from the same state and with the same symbol, the transition leads to different states. Yet this case is possible when clustering independently  $\mathbf{x}_i, \mathbf{s}_i, \mathbf{x}_{i+1}, \mathbf{s}_{i+1}$  into their own states as we do.

For iso-utility, iso-prediction and decisional states this is not a problem: As explained in Section 3.4 transitions are determined in terms of changes in utility related quantities, the string symbols are irrelevant in that case (in other words, the distinct causal states of the underlying  $\epsilon$ -machine would be merged into the coarser level). For an  $\epsilon$ -machine reconstruction however the proposed algorithm needs to be augmented with an additional step.

The user can optionally express symbol values together with each  $\mathbf{x}_i \rightarrow \mathbf{x}_{i+1}$  transition. These are used as constraints for the clustering algorithm when they are available. The following procedure is implemented:

- Clustering is performed as described in Section 5.4.
- After clustering, iterate the following steps:
  - Split step: It might be that data values  $\mathbf{x}^1$  and  $\mathbf{x}^2$  for the transitions  $\mathbf{x}^1 \xrightarrow{a} \mathbf{s}^1$  and  $\mathbf{x}^2 \xrightarrow{a} \mathbf{s}^2$  were clustered together, while  $\mathbf{s}^1$  and  $\mathbf{s}^2$  are not clustered together (see the above example). In that case, the state containing  $\mathbf{x}^1$  and  $\mathbf{x}^2$  is split in order to restore determinism.
  - Merge step: If several  $x_i = \mathbf{x} \xrightarrow{a} x_{i+1} = \mathbf{s}$  transitions are observed, with the same configuration value  $\mathbf{x} \in X$  and symbol  $a \in A$ , then all corresponding  $\mathbf{s} \in X$  are pre-clustered in the same state. Note that splitting acts on the states of mismatching configurations before a transition, while merging acts on the states of mismatches after a transition, so both can be applied without undoing each other.
  - Break the loop in the case of incompatible constraints and there is no convergence.

Convergence of the loop would effectively ensure determinism of the reconstructed automaton in the perfect case where all distribution estimations are exact.

Unfortunately this is not the case in practice. Indeed, clustering from finite data is necessarily imperfect. If  $x \in \sigma_2$  is wrongly affected to causal state  $\sigma_1$  then forcing symbol determinism might create spurious states:  $\sigma_2$  is erroneously split until the transitions are consistent, while the source of the inconsistency is not detected. Or similarly states are merged when they should not.

We had to accept a threshold for clustering distributions together (ex: a significance level for the Chi-Square test), due to the imperfect distribution estimation. In turn, we have no choice but to accept that some  $x \in \sigma$  might be misclassified and might generate spurious transitions. The same way we ignore small discrepancies in distribution clustering, the solution is to ignore small discrepancies in the automaton determinism. Formally:

Let  $\sigma$  be a causal state,  $a \in A$  a symbol in the alphabet  $A$ . The automaton is deterministic when each time a data value  $x \in \sigma$  is followed by the symbol  $a$  then  $s = xa$  falls in a unique causal state  $\varphi$ ,  $\forall x \in \sigma$ . When the automaton is not deterministic there is instead a distribution  $p(\Phi|\sigma, a)$  with  $\varphi \in \Phi$ .

We propose here to set a threshold  $\theta = 1 - \epsilon > \frac{1}{2}$  for ignoring small discrepancies up to  $\epsilon$ : When  $\exists \varphi / p(\varphi|\sigma, a) > \theta$  then the unique such  $\varphi$  is taken as the automaton transition. This threshold is completely separate from the probability of the transition itself.

Concretely, the split and merge step described above are applied only on such transitions  $\varphi$ , the spurious transitions are ignored.

## 5.6 Complexity of the algorithm

Depending on the user context, one or the other of these tasks might become the dominant algorithm cost:

- Estimating the probability distributions  $\hat{P}(Z|x)$ . For discrete data the computation cost is simply  $O(N)$ , a matter of counting the occurrences of each  $z \in Z$  for each  $x$ . But using the above Kernel Density Estimation the complexity is roughly  $O(N(M + q(N)))$  with  $N$  the data size and  $M$  the number of samples  $s \in S \subset Z$  at which  $\hat{p}(Z = s|x)$  is estimated.  $q()$  is the cost of performing a nearest neighbours query in the joint space (so negligible kernel values are quickly eliminated, the worse-case limit of  $q = O(N)$  is the summing of all kernel values at all data points). The nearest neighbors are estimated in a first pass in  $O(Nq(N))$  time and in the second pass the samples are used to build the distributions in  $O(NM)$  time.
- Clustering tasks. The two clustering algorithms described in Section 5.4 have a linear and a quadratic complexity with respect to the dissimilarity measure.

- Evaluating the utility function. For the analytical examples in the Section 4,  $U(y, z)$  is simple enough so its evaluation was not the main issue. However in a different scenario the algorithm complexity might have to be defined in terms of the number of evaluations of the cost function.
- Optimising  $\hat{U}(y|x)$  in order to find  $\hat{Y}(x) = \text{Argmax}_{y \in Z} \hat{U}(y|x)$ . An exhaustive search of  $Z$  is only feasible for small discrete spaces. Advanced multimodal optimisation techniques might become necessary and induce large computation times.

The memory requirements for running these computations might also become a limiting factor. For example it might not be possible to store all  $\hat{P}(Z|x)$  distributions for each unique  $x \in X$  present in the data set, especially if a large number of samples  $s \in Z$  is needed (ex: the Monte-Carlo sampling error decreases as  $O(1/\sqrt{|S|})$ ).

## 6 Application examples

The proposed algorithm is quite adaptable to various situations. Figure 4 actually defines a family of algorithms, depending on how the functional inputs are implemented. For example, estimating  $\hat{P}(Z|x)$  is quite difficult in general, and depending on the nature of the data (discrete or continuous) it can lead to many algorithm variants. Similarly for the clustering task. As the results are highly dependent on these crucial tasks, it is important specify these functional inputs together with the algorithm in concrete applications.

The following examples were thus chosen so as to highlight several cases:

- The core part of the algorithm, reconstructing the  $\epsilon$ -machine, with comparison to a classical benchmark in the literature [27].
- A toy application that is also compared to reference results in the literature [26]. The goals are to highlight how to use the light-cone formalism instead of symbolic series, as well as to show to use of a utility function.
- A larger application on a practical problem (edge detection) that highlights how to apply the algorithm on a larger scale, as well as the influence of the utility function.

In addition to these examples the algorithm was also applied to real electroencephalogram (EEG) data in a separate paper [8], which is out of scope of the present introductory material.

### 6.1 Reconstruction of the Even process

This first application example demonstrates the capability of performing an  $\epsilon$ -machine reconstruction. Since the

$\epsilon$ -machine is the minimal and optimal deterministic automaton for reproducing a process statistically, and since the decisional states transition graph is a sub-machine of the  $\epsilon$ -machine (see Section 3.5), the proposed algorithm needs to perform well on this task.

The Even process is used as a benchmark in [27]. A similar experiment is conducted with the proposed algorithm for comparison.

The Even process consists of two states, and generates binary strings where blocks of an even number of 1s are separated by an arbitrary number of 0s. Despite its apparent simplicity the Even process does not correspond to any finite state-output HMM [23], and requires the power of an  $\epsilon$ -machine to be reconstructed<sup>6</sup>. Figure 5 shows the process states and transitions.

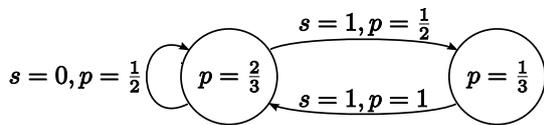


Figure 5: Definition of the Even process.

Data is generated according to the Even process as a series of symbols. The goal of the experiment is to reconstruct the underlying transition graph from these observations.

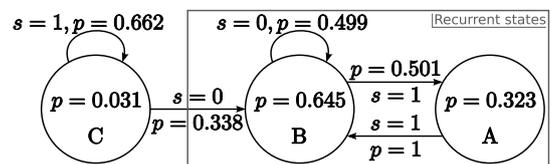
The algorithm described in Section 5 is set up with the following parameters:

- System configurations  $x \in X$  are taken as the symbols in a sliding window of size  $L$  past data values. The predictions  $z \in Z = \{0, 1\}$  are the symbol in the series following this window, matching what is used in CSSR.
- Discrete distributions are built by monitoring  $(x, z)$  pairs in the training set of size  $N$ .
- A Chi-Square test is used in order to match distributions, with 5% accuracy.
- The aggregative clustering algorithm described in Section 5.4 is applied, with a single pass.
- Symbol constraints are available and implemented as described in Section 5.5 with a tolerance threshold  $\theta = 0.95$ .

<sup>6</sup>Let's consider a naive state-output HMM emitting 0 in the left state of Fig. 5 and 1 in the right state. Let's start with the left state. There may be an arbitrary number of 0 emitted. When a transition is taken to the right state then a 1 is emitted after the transition. The process then returns to the left state and necessarily emits a 0. Adding a self-loop on the right state would not satisfy the even number of 1 requirement. We could chain two states emitting a 1 before returning to the left state, forcing the even number two. But then we would need another chain for emitting four symbol 1 in a row, and so on. In fact, no finite state-output HMM can be constructed for the Even process, while this is trivial with an edge-output HMM: that is, an  $\epsilon$ -machine.

- We are not interested in this example in decisional states, so we do not set a utility function.

The result of one reconstruction with a typical transient state, using  $N = 10^5$  associations and a past window of 10 points, is shown in Fig. 6. The recurrent causal states of this  $\epsilon$ -machine correctly correspond to the definition of the Even process. Close inspection of the data shows that the transient state corresponds to strings formed of 10 symbols 1 in a row. Due to the limit in window size the algorithm cannot distinguish whether the last symbol 1 was emitted from recurrent state A or B. Logically, it observes that  $\frac{1}{3}$  of the time the next symbol is a 0 in the data set and  $\frac{2}{3}$  of the time it is a 1, matching the proportions of the symbols in the data set:  $p(s = 1) = p(s = 1|A)p(A) + p(s = 1|B)p(B)$  as the process is really in either the state A or the state B.



Parameters:  $10^5$  data points, using a past window size of 10 points (random seed = 1).

Figure 6: Reconstruction of the Even process.

The proposed algorithm classifies every single training point in a causal state, hence creates transient states if necessary to match data. Note that despite the Even process not being equivalent to any finite state output HMM chain, the proposed algorithm reconstructs it fairly well with a window size of 10.

In [27, Fig. 4], an experiment is conducted to study the behaviour of the CSSR algorithm depending on the history size. The transposition of this experiment is conducted in the current framework in order to highlight the differences between both algorithm behaviours.

Figure 7 shows the result of that experiment: how many recurrent states are found on average (over 50 independent trials) by the proposed algorithm and by CSSR, depending on the window size. Results for  $L = 1$  and  $L = 2$  produce an incorrect transition graph, there is not enough history to reliably determine the states, and thus they are not presented in this figure.

Both algorithms reconstruct the states correctly when there is enough data compared to the history length. Their behavior differs when the number of observations is not enough to estimate the distributions of histories correctly. This happens both when  $N$  is too small (not enough observations), and when  $L$  is too large (too many possible histories).

CSSR may split the states with each increase in the window size, whereas the present algorithm clusters states using the whole window and symbol constraints. When there is not enough data to estimate the distributions properly

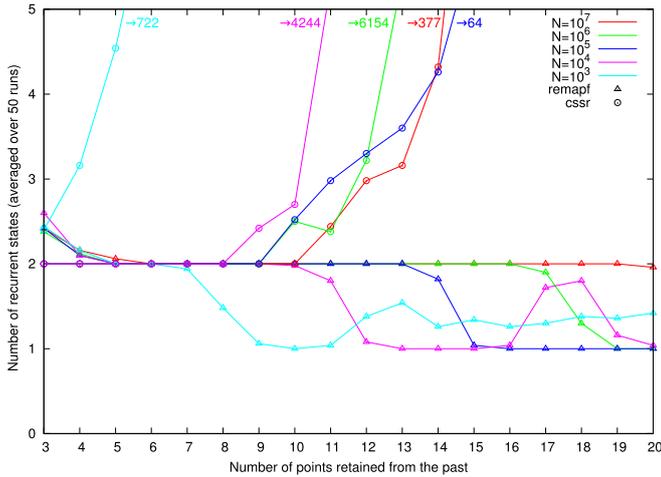


Figure 7: Number of recurrent states reconstructed from the Even process by the REMAPF and CSSR algorithms.

CSSR over-splits the states. The current algorithm merges them.

The proposed algorithm is also more robust, at least on this example. Fig 7 shows that for each data size  $N$  the REMAPF algorithm can correctly reconstruct the states for a larger range of history size  $L$  than CSSR. The larger  $N$ , and the larger that range of histories.

Regarding the computation time, CSSR worst case complexity [27] is  $O(n_e^{2L+1}) + O(N)$  where  $n_e$  is the number of edges per node in the reconstructed graph (hence  $n_e$  is at most the alphabet size  $|A|$ ). On average CSSR is faster than its worst case, but still exponential in history length. REMAPF is constant time with respect to the history length. With the chosen clustering algorithm in this section REMAPF is  $O(KN)$  where  $K$  is the number of estimated states. For small history lengths CSSR is faster than REMAPF as  $O(n_e^{2L+1}) + O(N) \approx O(N)$  in that case, while REMAPF remains  $O(KN)$ . Fortunately REMAPF does not oversplit the states, hence its computation time remains small in the pathological cases, while CSSR computation time explodes. Table 1 shows the computation times averaged over 50 runs for both algorithms and at each history size  $L$  (thus 50 runs per table cell, averaged), computed on a 2.2 GHz machine.

## 6.2 Cellular automaton

Another test case where causal states were applied is the detection of moving particles in cellular automata, and their interactions [26]. The introduction of a utility function in this context provides a simple yet effective way to demonstrate the concepts presented in this document. The next section considers the usage of decisional states in a larger-scale application.

Figure 8 shows the evolution rule and configuration of an elementary one-dimensional automaton. Each row of

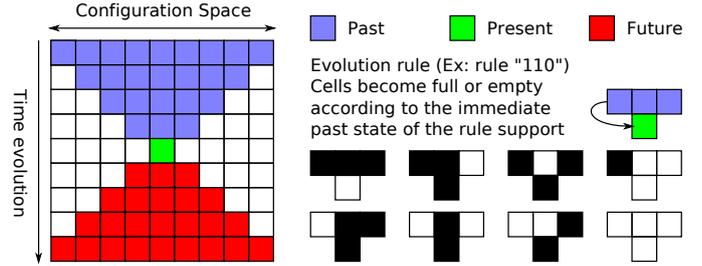


Figure 8: Elementary cellular automaton.

the regular grid contains the system state at a given time. An evolution rule dictates how the cell binary states evolve at each time step. The numbering scheme “110” refers to an exhaustive listing of all possible elementary cellular automata rules [34]. The evolution rule has a support, 3 cells in figure 8, from which the next cell configuration is deduced. Propagation of this support in time defines “light-cones” according to the terminology of Appendix A, within the implementation constraint of a limited depth.

In this context the data set  $X$  is the space of all past light-cones (in blue on Figure 8). From the current system state we would like to predict the future of the system, so  $Z$  is the space of all future light-cones (in red on Figure 8). Even though the cellular automaton is completely deterministic, the state of cells in the future cone depend on information which is outside the past cone, so we observe a distribution of different futures for each past (see Section 2.1). With cyclic boundary conditions and a fixed evolution rule for the whole automaton, all cells have exactly the same distribution so we can aggregate the observations across all cells.

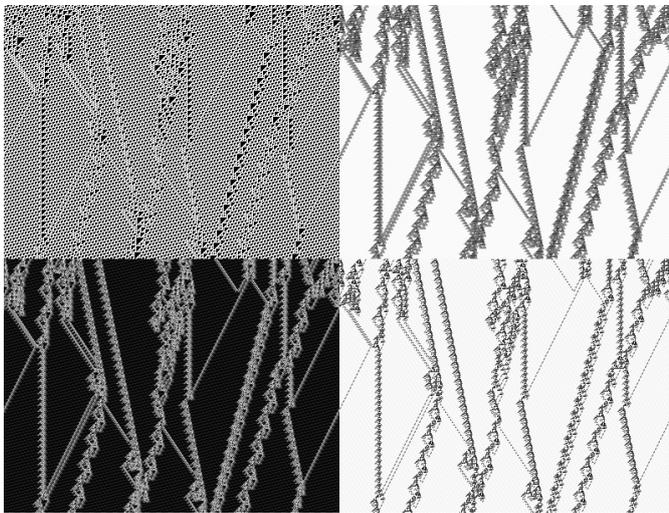
The utility function is chosen by the user according to the application needs. Here we chose to define the utility of a prediction as the number of correctly predicted cell states in the future cone. Hence the utility takes in this example integer values between 0 and the maximum  $d^2 - 1$  where  $d$  is the future cone depth (we could also have used a proportion between 0 and 1).

Given the discrete nature and relatively small search space of the problem, the algorithm described in Section 5.1 is setup with:

- A simple discrete probability density estimator based on  $(\mathbf{x}, \mathbf{z})$  observation counts:  $\hat{p}(\mathbf{z}|\mathbf{x}) = \frac{\text{count}(\mathbf{x}, \mathbf{z})}{\text{count}(\mathbf{x})}$ .
- An exhaustive integrator weighting the utility of all possible future cones by their probability:  $\hat{U}(y|\mathbf{x}) = \sum_{z \in Z} \hat{p}(z|\mathbf{x})U(y, z)$ . In practice unobserved  $z$  values would induce a null contribution so the summing occurs only on observed  $z$ .
- An exhaustive search optimiser, computing  $\hat{U}(y|\mathbf{x})$  for all possible  $y \in Z$ . The maximum utility value as well as the set  $Y(\mathbf{x}) = \text{Argmax}_{y \in Z} \hat{U}(y|\mathbf{x})$  of best predictions are maintained during the search.

N	Algo	L=3	L=4	L=5	L=6	L=7	L=8	L=9	L=10	L=11	L=12	L=13	L=14	L=15	L=16	L=17	L=18	L=19	L=20
1e3	R	0.01	0.01	0.009	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.01	0.02	0.01	0.01	0.01
	C	0.01	0.01	0.009	0.01	0.01	0.01	0.02	0.02	0.04	0.08	0.14	0.20	0.29	0.34	0.40	0.49	0.58	0.68
1e4	R	0.02	0.03	0.02	0.03	0.02	0.03	0.02	0.03	0.03	0.03	0.04	0.04	0.06	0.1	0.14	0.21	0.26	0.27
	C	0.01	0.01	0.01	0.02	0.01	0.02	0.02	0.02	0.03	0.05	0.13	0.48	1.52	4.3	10.5	22.5	41.6	72.1
1e5	R	0.16	0.17	0.19	0.17	0.19	0.2	0.19	0.18	0.21	0.21	0.20	0.21	0.29	0.36	0.55	1.02	2.17	5.03
	C	0.04	0.04	0.04	0.05	0.05	0.05	0.05	0.06	0.06	0.08	0.11	0.20	0.44	1.7	7.82	28	120	512
1e6	R	1.5	1.49	1.5	1.54	1.53	1.54	1.55	1.62	1.6	1.64	1.73	1.74	1.94	1.95	2.24	2.95	4.74	14.4
	C	0.3	0.31	0.33	0.35	0.36	0.38	0.39	0.41	0.44	0.47	0.53	0.64	0.87	2.1	7.45	23.4	69.5	217
1e7	R	14.7	14.7	14.9	15.1	15.1	15.2	15.3	15.8	15.8	16.2	16.4	17	17.6	18.1	19.2	22.4	26.9	40.7
	C	2.89	3.11	3.25	3.41	3.54	3.6	3.74	3.92	4.1	4.32	4.6	4.94	5.45	6.97	12.7	29.2	74	200

Table 1: Timings (seconds) for the experiments in Fig.7



Top-left: Cellular automaton cell values (1 is white, 0 is black).  
 Top-right: Local statistical complexity field (difficulty to get the future distribution) scaled between white (minimal value) and black (maximal complexity).  
 Bottom-left: Local expected utility field (not the complexity, the value of the expected utility itself) scaled similarly.  
 Bottom-right: Local iso-prediction complexity field (difficulty to get an optimal prediction) scaled similarly.  
 Parameters: Past depth 4, future depth 3, 400 cells, 300 steps, 100 initial transient dropped.

Figure 9: Raw cells and complexity fields of a cellular automaton.

- The connected component, single-link hierarchical clustering algorithm described in Section 5.4, with exact match predicates.

Figure 9 shows the results of the experiment for the evolution rule introduced in Fig. 8. Figure 9 is directly comparable with [26, Fig. 3], where another algorithm was used to estimate the  $\epsilon$ -machine. The raw cellular automaton field is the direct application of the rule depicted in Fig. 8. The statistical and iso-prediction complexity fields are mapped to a grey scale range where white represent

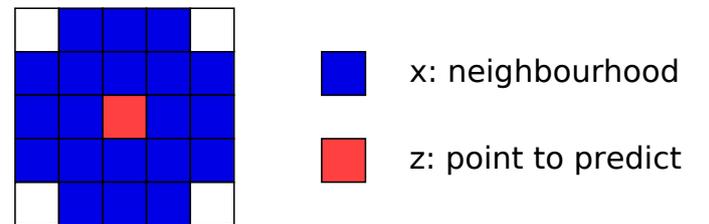


Figure 10: Neighbourhood for image filtering.

their respective minimum complexity value and black their respective maximum. The expected utility field is also shown in 9.

The causal states sub-cluster the iso-predictive states: we observe alternative versions of the particles. The utility is based on the number of correctly predicted cells, irrespectively of their position in the future cone. Information that is irrelevant to this utility function is masked out in the iso-prediction field, whereas it was present in the statistical complexity field.

Some information was lost. But if all the user cares is encoded in the utility function, that information was noise and clarity was gained in the result. In extension to [26], we have defined a new family of automatic filters based on utility functions.

### 6.3 Image filtering and edge detection

The idea of this section is to extend the cellular automaton example for filtering images. We make the hypothesis that edges correspond to zones where the prediction difficulty is greatest. This differs from other common definitions, like a high luminance gradient magnitude. The definition of an edge is not the topic of this article. This section's goal is to show how the concepts introduced in this document might be used on a concrete non-temporal data example.

Following the construction in [1], the data space  $X$  is defined as the neighbourhoods of image pixels  $z \in Z$ . The prediction problem is to find the value of  $z$  from the neighbourhood. Figure 10 shows how the neighbourhood is

defined in this experiment: up to two pixels in each direction except corners. Larger or smaller neighbourhoods were tested: smaller regions make the prediction more difficult, larger regions lead to thicker detected edges. Similarly defining  $z$  as a centre block instead of a single pixel was also tried, with similar observed effects (precision and edge size). For a 8-bits grey image the data space is thus  $X = [0 \dots 255]^{20}$  and the prediction space  $Z = [0 \dots 255]$ .

The data space  $X$  is thus considerably larger than in the previous examples. Fortunately, unlike the cellular automaton case where a difference in  $x$  can lead to completely different outcomes, usually images are not significantly altered when pixel values differ by a small amount. In the present context we exploit this nearby consistency in  $X$  and  $Z$  in order to apply kernel density estimators. These are more reliable than the simple count-based estimator used in the previous section, especially since  $X$  has a higher dimension.

The approach of considering the image as the limit distribution of a Markov Random Field [1] is applied to this example. Concretely, this amounts to estimating the probability densities in the joint space  $X \times Z$  and inferring the conditional distributions by integration of  $\hat{P}(X)$ , as described in Section 5.2.

The prediction space  $Z$  can be run through exhaustively in the case considered here: only one centre grey pixel. Sub-sampling for numerical integration is thus not necessary, and we set  $S = Z$ .

The utility of a prediction  $y$  when the true value  $z$  happens is defined as  $U(y, z) = -\max(0, |y - z| - \tau)$ . In other words small prediction discrepancies up to  $\tau$  are accepted at no cost, reflecting the fact the image is not significantly altered by small variations in pixel grey levels. Then the utility decreases (the loss increases) with each grey level difference between the predicted and the true value.

A pre-processing is performed. For each region  $x$  the minimal grey level is computed. It is then subtracted from both  $x$  and  $z$  without loss of genericity (the original grey level for a prediction  $z$  can be reconstructed by adding back the value shift defined on  $x$ ). A post-processing is performed: ordering the states by their complexity values and plotting their ranks with respect to that ordering on a grey scale. Fig. 11 shows the result of this experiment.

The effect of applying a utility function is apparent on the right part of Fig. 11. The noise inherent present in the image filtered by causal states may be attributed to errors in the density estimation and clustering technique: some extra states that appear only once, thus with high complexity value (black dots). When applying a utility function these extra states are merged into coarser states with the same properties with respect to that utility function. The number of black dots thus decreases. The utility function also manifests itself in the reduction of the background noise in the image. Since a tolerance is given the small differences in grey levels are ignored (up to  $\tau = 7$

in this example at no cost). The flat zones in the picture are thus more uniformly white (ex: the shoulder at the middle-bottom of the picture).

The filter defined by local complexity values has several characteristics that differ from other more traditional image processing techniques:

- The filter is defined globally: features are defined on the whole picture. For example the flat zones in the original image are detected as having low complexity. When the filter is applied locally this information is taken into account. The extreme example would be the cellular automaton background in Figure 9: many edges would be detected for each small triangle pattern using a simple gray level gradient based filtering, but the proposed filter assigns a low value for these repetitive patterns.
- Fine details are similarly considered statistically on the whole picture. In the bottom-left region of the picture, the filaments attached to the hat are detected as single units: each light-dark transition has its own complexity, low values are whitened out by the ranking. This also differs from gray level gradient filtering techniques that are prone to emit an edge on each side of the filament.
- Making global statistics and clustering probability distributions comes with a computational cost. The time consuming task is to build the statistics: it took 37 minutes on a 2.2Ghz quad-core CPU to build the probability distributions for the Fig. 11 with a kernel width of 1.5 units in the joint space  $X \times Z$ . By comparison clustering the causal states with 3 passes of the linear clustering took 57 seconds, and applying the utility took 100 ms.

It would be interesting to combine the complexity-based filters to other well-established edge detection, segmentation or noise removal algorithms. The goal of this article is to introduce the decisional states and their applications in different contexts. The proposed edge detection is only a demonstration of how the main concept might be used in practice, this framework may possibly be adapted for generic feature detection with a variant setup (especially alternative utility functions and larger support for the pixels in the  $X$  space).

## 7 Conclusion

The decisional states notion was introduced: the internal states of a system that lead to the same decision, given a user-defined utility function. Compared to alternative approaches in the domain [15, 33], here the utility function is defined on the space of predictions:  $U(y, z)$  quantifies what gain/loss is incurred when  $y$  is predicted while  $z$  happens. This makes the present work suited for applications like



Left: Filter obtained with local statistical complexity ranks (see main text)

Middle: Original Lenna Image

Right: Filter obtained with local iso-prediction complexity ranks. The background noise is eliminated but the edges are preserved.

Figure 11: Proposed Image filter (top left) and some variants.

time series processing and detection of anomalous/more complex zones in a system, while less suited for reinforcement learning [30].

The natural framework for applying the utility function is the  $\epsilon$ -machine [12], which is an edge-emitting Markovian graph model with higher genericity than the usual state-output Markov chains [23]. In this context the  $\epsilon$ -machine corresponds to the internal structure of the system, irrespectively of any user-defined utility. The decisional states are built on top of this internal structure in a way that reflects the external knowledge (encoded in the utility function) brought in the system.

Coming with the decisional states are definitions of complexity measures on the system. It is possible to quantify precisely, in number of bits, the difficulty of making an optimal prediction in terms of the chosen utility. Another consequence is a way to identify events that provoke a change of decision, represented as transitions in a state diagram, assuming decisions are based on the expected utility. The decisional states were exemplified mathematically on analytically tractable examples, and numerically on practical problems like image filtering. A separate article [8] shows how to use this notion on large data sets of real EEG signals.

A new algorithm was introduced for computing an  $\epsilon$ -machine from observed data and for computing the newly introduced decisional states on top of it. This algorithm is very adaptable to specific application needs, including for temporal and spatial data, using a symbolic representation or not, as demonstrated by the examples in the previous sections. A reference implementation is provided, see the Appendix. It is available as free-libre software.

## Appendix: Web information

The latest experimental version of the code as well as previous versions are available at the source repository at [http://source.numerimoire.net/decisional\\_states](http://source.numerimoire.net/decisional_states)

The code is highly templatised and the classes might be directly included into a user project without having to link to an external library. The code is available as free-libre software (GNU LGPL v2.1 or more recent) and contributions are welcome.

## References

- [1] S. P. Awate. *Adaptive Nonparametric Markov Models and Information-Theoretic Methods for Image Restoration and Segmentation*. PhD thesis, University of Utah, USA, 2007.
- [2] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, New York, 1985.
- [3] G. Bertello, P.-J. Arduin, F. Boschetti, and D. Weatherley. Application of computational mechanics to the analysis of seismic times-series via numerical optimisation. *New Generation Computing*, 27:1–23, 2009.
- [4] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):192–236, 1974.
- [5] C. Bishop. *Pattern Recognition and Machine Learning*. Springer Verlag, 2006.

- [6] N. Brodu. Quantifying the effect of learning on recurrent spiking neurons. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2007.
- [7] N. Brodu. A synthesis and a practical approach to complex systems. *Complexity*, 15(1):36–60, 2008.
- [8] N. Brodu, F. Lotte, and A. Lécuyer. Exploring two novel features for eeg-based brain-computer interfaces: Multifractal cumulants and predictive complexity. Submitted preprint, <http://arxiv.org/abs/1003.2879>, 2011.
- [9] P. Bühlmann and A. J. Wyner. Variable length markov chains. *The Annals of Statistics*, 27(2):480–513, 1999.
- [10] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 2:1023–1028, 1994.
- [11] G. Celeux and J.-B. Durand. Selecting hidden markov model state number with cross-validated likelihood. *Computational Statistics*, 23(4):542–564, 2008.
- [12] J. P. Crutchfield and K. Young. Inferring statistical complexity. *Physical Review Letters*, 63(2):105–108, 1989.
- [13] T. Duong. *Bandwidth selectors for multivariate kernel density estimation*. PhD thesis, University of Western Australia, 2004.
- [14] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceeding of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996.
- [15] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
- [16] H. Jaeger. Observable operator models for discrete stochastic time series. *Neural Computation*, 12(6):1371–1398, 2000.
- [17] H. Jänicke, A. Wiebel, G. Scheuermann, and W. Kollmann. Multifield visualization using local statistical complexityalexander wiebel. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1384–1391, 2007.
- [18] D. H. Johnson and S. Sinanović. Symmetrizing the kullback-leibler distance. Unpublished manuscript, 2001.
- [19] M. L. Littman, R. S. Sutton, and S. Singh. Predictive representations of state. *Advances in Neural Information Processing Systems*, 14:1555–1561, 2002.
- [20] W. Löhr. *Models of Discrete-Time Stochastic Processes and Associated Complexity Measures*. PhD thesis, University of Leipzig, 2010.
- [21] W. Löhr and N. Ay. On the generative nature of predictions. *Advances in Complex Systems*, 12(2):169–194, 2009.
- [22] R. A. McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the twelfth International Conference on Machine Learning*, pages 387–395, 1995.
- [23] A. Ray. Symbolic dynamic of complex systems for anomaly detection. *Signal Processing*, (84):1115–1130, 2004.
- [24] C. R. Shalizi. *Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata*. PhD thesis, University of Wisconsin at Madison, USA, 2001. PhD dissertation.
- [25] C. R. Shalizi. Methods and techniques of complex systems science: An overview. *Complex systems science in biomedicine*, pages 33–114, 2006.
- [26] C. R. Shalizi, R. Haslinger, J.-B. Rouquier, K. L. Klinkner, and C. Moore. Automatic filters for the detection of coherent structure in spatiotemporal systems. *Physical Review E*, 73:036104, 2006.
- [27] C. R. Shalizi and K. L. Shalizi. Blind construction of optimal nonlinear recursive predictors for discrete sequences. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*, pages 504–511, 2004.
- [28] J. S. Simonoff. *Smoothing Methods in Statistics*. Springer-Verlag, New York, 1996.
- [29] S. Still. Information theoretic approach to interactive learning. *EuroPhysics Letters*, 85(2):28005, 2009.
- [30] P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *Proceedings of the ICML’04 workshop on Relational Reinforcement Learning*, 2004.
- [31] E. N. Talvitie. *Simple Partial Models for Complex Dynamical Systems*. PhD thesis, University of Michigan, 2010.
- [32] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Elsevier, third edition, 2006.
- [33] D. Wierstra and M. Wiering. Utile distinction hidden markov models. In *Proceedings of the twenty-first International Conference on Machine Learning*, 2004.
- [34] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, 1983.