What should I notice? Evaluating memorability of events to generate surprising causal candidates.

*† Étienne Houzé, * Jean-Louis Dessalles, * Ada Diaconescu, † David Menga
* Télécom Paris, IP Paris, *Palaiseau*, France
email: {first}.{second}@telecom-paris.fr
† EDF R&D, *Palaiseau*, France
email: {first}.{second}@edf.fr

Abstract—**When confronted to an unprecedented situation, humans typically show good performance in quickly identifying noticeable past events and proposing them as possible causal hypotheses. This kind of abductive inference is widely overlooked in modern AI approaches which rely on massive datasets to learn associative patterns. Our proposal is to formalize and compute a "memorability" score over a memory of various recorded events from a cyber-physical system. This score can later be used either to select only relevant information to be remembered, or to propose causal hypotheses in unusual situations, on demand. As such, the approach aims at being complementary to more traditional learning-focused techinques. We provide theoretical ground for our approach by using and extending existing results and ideas from Algorithmic Information Theory and provide an implementation example, showing practical results in a smart-home scenario.**

Index Terms—**Complexity, Algorithmic Information Theory, Simplicity, Abduction, Surprise**

I. Introduction

The ability to infer new causal relations is one of the many tasks in which humans prevail but machines still fall short of completing. This ability is prime in making discoveries, or acquiring new knowledge by making educated guesses.

Without any knowledge about how physics or electricty work, it would be possible to infer a possible link between a memorable thunderstorm and a general black-out only by using the noticeability score of the first. While this kind of abductive inference can yield many false-positives, its application can be to propose new hypotheses to be tested out when other possible methods of inference fail to provide any hypothesis (due to a lack of prior similar situations, or to the importance of the context). The difficulty of this approach to reasoning is, however, to quickly identify the relevant candidate hypotheses. In this regards, we may use a basic instinct: without prior knowledge, the most relevant hypothesis might simply be the most memorable recent event. However this definition is highly subjective, and does not seem to fit well into the canvas of computing and AI.

The difficulty of this computation comes from different factors. First, events can be of different nature, and not directly comparable. In fact, for the use case of smart homes, or other cyber physical systems, events range from device removal to a detected presented or an unusually high temperature. How can one can then assess which ones are more memorable? Furhtermore, even for comparable events, different characterics can be put forward to argue for the most memorable: is a record-high temperature 47 days ago more memorable than the small deviation recorded just 3 minutes ago? To this day, no current system proposes to use all this information from various event types from different devices to compute a unified metrics of "memorability".

To tackle this challenge, we propose to refer to events using a technique used by humans: naming. It appears that humans are able to name events and describe them, regardless of their nature or their characteristics. Furthermore, there is a noticeable correlation between the lenght of the description, in terms of the complexity of concepts and arguments, and the "memorability" of the event. Think, for instance, of how "last year's hottest day" appears simpler than "the 182th day of 7 years ago". Using this method, the day described by the first sentence would be deemed simpler than the target of the second description. By comparing the description length of the event with the average description lenght of similar events, one would then have a measure of the "memorability" of events, as memorable events stands out as much simpler (or more complex) than their neighbours.

To programatically achieve what humans are easily capable to do, we rely on a basic model intuition of how memory works. Figure 1 shows this process for the example description "last year's hottest day". We assume that a memory $\mathcal{M}$ is a unordered set of various distinct objects, upon which can be applied boolean predicates π to describe them. These predicates thus provide filters f_π that can be applied to the memory, selecting all events satifying predicate π. By applying successive filters, we narrow down the size of the memory, until a single event remains. The complexity of this remaining events can thus be estimated as the simplest combination of predicative filters used to retrieve it.

The rest of this paper is organized as follows: we first briefly introduce some notions of Algorithmic Information Theory and its applications in section II, then we explicit our methods of computing the unnexpectedness of past events in cyber-physical systems III. We illustrate this approach by providing an implementation relying on a few predicates on events from a smart home simulation (section IV). In section V we briefly review other related works using complexity theory or trying to explain smart homes and how our work can be linked with them before exploring potential extensions of our work in

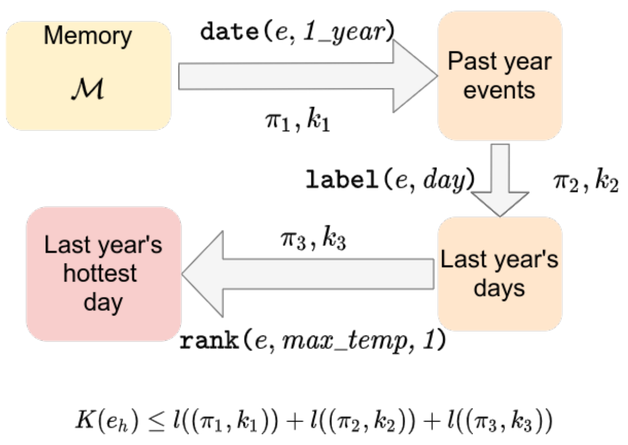


Fig. 1. An example of retrieving an event through successive predicative filters. From the base memory (yellow), successive filters select the events satisfying their predicates (grey arrows). The operations yields successive subsets of the memory (yellow), until a final singleton memory is reach, meaning that the successive filters retrieve a unique event from the memory. The predicates used for this retrival path and their arguments, $\pi_1, \pi_2, \pi_3, k_1, k_2, k_3$, can be used as a retrieval program. Its length gives an upper bound to the description complexity of the event e_h: "last year's hottest day".

section VI

II. Theoretical Background

Kolmorov complexity formally quantifies the amount of information required for the computation of a finite binary string[1] (or any object represented by a finite bynary string) [1], [2]. For such a binary string s, its complexity $K(s)$, is the bit length of the shortest program p which, if given as input to a universal Turing Machine U, outputs s.

$$K_U(s) = \min_p \{l(p) | U(p) = s\} \tag{1}$$

The first notable property of this definition is its universality: while the choice of the Turing machine U used for the computations appears in the definition of eq. 1, all results stand, up to an additional constant. Think, for instance, how any Turing-complete programming language can be turned into any other language, using a compiler program. In fact, since any Turing machine U' can be emulated by U from a finite program p_U, we have the following inequality:

$$K_U(s) \leq l(p_u) + K_U(s) \tag{2}$$

From this first result, we can then define a universal complexity $K(s)$, which no longer depends on the choice of a Turing machine U, such that, for any machine U,

$$\forall U \in \text{TM}, \forall s, |K(s) - K_u(s)| \leq C_U \tag{3}$$

where the additional constant C_U does not depend on the object s.

[1]While the defintion still holds for some inifinite binary strings (think of the representation of the decimals of π), we restrict ourselves to finite strings in this paper.

It is important to note that the notion of Kolmogorov complexity is not related to the computation complexity, as there is no requirement on the execution time of the programs, only their length in bits matters for the computation of complexity. In fact, it can be shown that Kolmogorov complexity is not computable [2]. The sketch of the proof is the following: since the definition of complexity from eq. **??** requires to find the shortest of all programs on a given Turing machine outputting s, it requires to run all programs up to a given length and compare their result to s. However, being able to do so means that we would be able to tell whether these programs terminate. Since the termination of programs is famously undecidable, by extension, the computation of Kolmogorov complexity is impossible. One can however easily approximate it with upper bounds, by exhibiting a program outputting s.

While the introducion of Kolmogorov complexity was purely theoretical, many applications were found, notably because of its close relation with the intuitive notions of the complexity of items, events, situations. For instance, [3] studied the use of Kolmogorov Complexity to identify and solve analogies.

Algorithmic Information Theory is also used to define the notion of unexpectedness: in [4], the unexpectedness $U(e)$ of an event is defined as the difference between an a-priori expected causal complexity $C_w(e)$ and the actual observed complexity $C(e)$.

$$U(e) = |C_w(e) - C(e)| \tag{4}$$

This result comes from the understanding that, while Kolmogorov complexity is computed with Turing machine, it can be used as an elegant proxy for modeling information processing in the human brain, and thus helps design a notion of simplicty or complexity of events [].

Using this measure, it is possible to model phenomenons such as coincidences: imagine that you happen to run into a friend in a park. The unnexpectedness will be much larger if this park is in Singapore than if it is in your common neighbourhood: the causal complexity C_w is much higher if you a priori have no knowledge of your friend being around, while the description complexity remains the same.

III. Computing the unexpectedness of events

A. A memory model

We model the memory as an unordered set M of events e. Individual events can be seen as data points, augmented with a label indicating their nature (temperature events, failure events, addition/removal events), and a timestamp:

$$e \equiv (l, t, \mathcal{D}) \tag{5}$$

where l is the label, t a timestamp and $\mathcal{D}$ a multi-dimensional datapoint representing various characteristics of the events (e.g. its duration, the maximum temperature reached, the sensor name, its position).

We define *predicates* as boolean function operating on these events, and taking an additional argument, encoded as a finite binary string:

$$\pi : \begin{cases} \mathcal{M} \times \{0,1\}^* & \mapsto \{0,1\} \\ (e,k) & \mapsto \pi_k(e) \end{cases} \tag{6}$$

An example of predicate is to use $\pi = \texttt{year}$ and $k = 1$, thus constructing the predicate $\texttt{year}()e, Y)$, which tells whether the event e occurred Y years ago.

From these predicates, we can construct filter operations on the memory. A filter operates by selecting the events satisfying a given predicate π_k and constructing a resulting memory from these events.

$$f_{\pi,k}(\mathcal{M}) = \{e \in \mathcal{M} | \pi_k(e)\} \tag{7}$$

For instance, using the same $\pi = \texttt{year}$ and $k = 1$ as above, we would build the filter $f_{\pi,k} = \texttt{last\_year}$, which selects all events that occurred last year.

B. Description complexity of events

From the previous definitions, we then estimate the complexity of events as the "simplest" way of retrieving them from the memory, by applying successive filters. A retrieval path composed of various filters $p = f_1 = f_{\pi_1,k_1}, f_2 = f_{\pi_2,k_2}, \dots, f_n = f_{\pi_n,k_n}$ is said to *retrieve* an event e from a memory $\mathcal{M}$ if and only if $p(\mathcal{M}) = f_1 \circ f_2 \circ \cdots \circ f_n(\mathcal{M}) = \{e\}$. For an event e, if such a path exist, its complexity can be bounded by $K(p) = \sum_{f_{\pi,k} \in p} l(\pi) + l(k)$, the complexity of the retrieval path being the sum of the bit lengths required to describe the predicates π and their arguments k.

The description complexity of an event e can thus be defined as the minimum complexity of a retrieval path retrieving e:

$$K(e) = \min_{p | p(\mathcal{M}) = e} K(p) = \min_{p | p(\mathcal{M}) = e} \sum_{f_{\pi,k} \in p} l(\pi) + l(k) \tag{8}$$

The definition used for description complexity in eq. 8 allows for a direct implemenation, which is shown in alg. 1. This algorithm operates iteratively: starting with the base memory $\mathcal{M}$ (line 1), we apply all possible predicate concepts π and programs k (lines 6-7), up to a given length `max_len` bits, and apply them: $M' = f_{\pi,k}(M)$ (line 12). We then store the pairs $(M', \texttt{len}(\pi, k))$ in an array $\texttt{future}_{\texttt{explore}}$. At the end of the iteration, the results of the filters become the memories which will be explored during the next iteration (lines 21–23). Each pass thus explore retrieval paths of increasing length. When a singleton memory is reached, the complexity of its unique element is upper-bounded with the length of the corresponding retrieval path (line 14).

C. Computing an unexpectedness measure

Equation 4 gives a universal definition of unexpectedness, as the difference between the complexity of the phenomenon generating an event and the complexity required to describe this event. In our particular application case, we may directly use the description complexity $K(e)$ as defined in 8.

Algorithm 1: Iterative computation of the complexity

```
1  $\texttt{current}_{\texttt{explore}} \leftarrow [\mathcal{M}]$ ;
2  $\texttt{future}_{\texttt{explore}} \leftarrow [\,]$ ;
3  $\texttt{pass} \leftarrow 0$ ;
4  while $\texttt{current}_{\texttt{explore}} \neq [\,]$ and $\texttt{pass} < \texttt{max\_pass}$ do
5      for $(\texttt{M}_{\texttt{prev}}, \texttt{K}_{\texttt{prev}}) \in \texttt{current}_{\texttt{explore}}$ do
6          for $\texttt{p} \in \mathcal{P}$ do
7              for $\texttt{k} \in \{0,1\}^*$ do
8                  $\texttt{K}_{\texttt{current}} \leftarrow \texttt{l(p)} + \texttt{l(k)} + \texttt{K}_{\texttt{prev}}$ ;
9                  if $\texttt{K}_{\texttt{current}} > \texttt{max}_{\texttt{complex}}$ then
10                     break ;
11                 end
12                 $\texttt{M}' \leftarrow f_{p,k}(\texttt{M}_{\texttt{prev}})$ ;
13                 if $\texttt{M}' = \{\texttt{e}\}$ then
14                     $\texttt{K(e)} \leftarrow \min(\texttt{K(e)}, \texttt{K}_{\texttt{current}})$ ;
15                 else
16                     $\texttt{future}_{\texttt{explore}}.\texttt{append}((\texttt{M}', \texttt{K}_{\texttt{current}}))$;
17                 end
18             end
19         end
20     end
21     $\texttt{current}_{\texttt{explore}} \leftarrow \texttt{future}_{\texttt{explore}}$ ;
22     $\texttt{future}_{\texttt{explore}} \leftarrow [\,]$ ;
23     $\texttt{pass} \leftarrow \texttt{pass} + 1$;
24 end
```

The term $C_w(e)$ from eq. 4 is harder to formally fit into our canvas. As C_w is defined as the shortest program a "W-machine" replicating the world's behaviour, based on the person's knowledge [4], it is convenient to understand this term as the expectation one would have regarding the complexity of the event e, *before* actually evaluating it.

A possible proxy for this quantity is to compute the average complexity of *similar* events. Still, difficulty remains in the definition of what can be considered *similar* events. Given that we have already used the notion of predicates to describe the events and compute their description complexity, we can use them as simple metrics for similarity: for each predicate π satisfied by e, we can define a set of similar events $N_\pi(e)$ as all the events also satisfying π.

$$N_\pi(e) = \{e' \in \mathcal{M}, \pi(e') \wedge e' \neq e\} \tag{9}$$

Now, when considering, for all possible predicates π, the corresponding neighbourhoods $N_\pi(e)$, with the convention that $N_\pi(e) = \emptyset$ if $\pi(e) = \perp$, we can compute an average expected complexity for e:

$$K_{exp}(e) = \frac{\sum_\pi \sum_{e' \in N_\pi(e)} K(e')}{\sum_\pi |N_\pi(e)|} \tag{10}$$

The definition used for the expected complexity of the event e in eq. 10 holds the intuition of favorizing more similar events: if e' is *very* similar to e, it will appear in many neighbouhoods, since it satisfies mostly the same predicates as e. Therefore, its complexity term will be present in more terms in eq. 10, and will weight more towards the final result.

D. *From memorability to abduction*

The problem of the abductive inference is different from the basic computation of the memorability score. Here, *knowing* that we want to find a cause c to an observed effect e, we try to find the most remarkable event in past memory in that regards. While our "memorability" score aims to identify the most remarkable events, it does not take into account the added knowledge about the effect e that is available for abductive inference.

This added knowledge would however influence the unexpectedness of a possible cause c, from eq. 4: even though the causal complexity $C_W(c)$ is likely to remain unchanged, as it relies on the model of the world in the person's mind, the description complexity $C(c)$ is replaced with the conditional complexity $C(c|e)$, as the knowledge of e is included in the problem's data.

In our canvas of predicative filtering, this corresponds to adding for free, in terms of program length, information about e to predicates. This translates into considering relative order between events: when investigating a cause for an anomaly in the living-rooms, other anomalies occurring in the same living-room will be simpler, since the location "living-room" is already known from knowing the consequnce. In terms of predicates, we therefore append a description of e to all programs k of predicates: $\pi_k(c) = \pi_{k::e}(c)$, where :: is the append operation.

This leads to another score $Abd(c, e)$ (abduction score of effect c for cause e), which is defined similarly to the description complexity in eq. 8 and using relative filters.

$$Abd(c) = \min_{p|p(\mathcal{M}|e)=c} K(p) = \min_{p|p(\mathcal{M}|e)=c} \sum_{f_{\pi,k} \in p} l(\pi) + l(k) \tag{11}$$

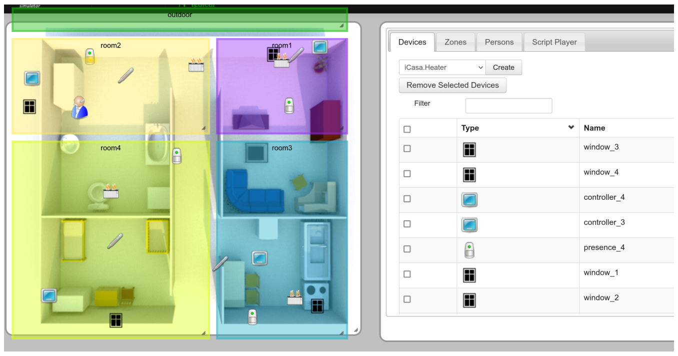
This abduction score translates the additional information provided to the system when answering a user's request. As such, it encapsulates the idea presented as the motivation of this paper: when confronted to a surprising situation, and with no other source of information, it can be used as a metrics to find relevant posisble surprising causes to the observed consequence.

IV. Implementation Example

A. *Setup*

To test our approach, we set up an experimental smart home setup. This choice of configuration is motivated by the challenges posed by smart homes for abductive inference: i) as the number of connected devices increases, more events are recorded, making the detection of memorable events more important; ii) smart homes are prone to undergo atypical situations, highly dependant on the context, for which pre-established relations might fail to find good abduction candidates. Furthermore, the choice was also motivated by the ease of application: as previous works exist on smart homes, it is possible to simulate their behaviour and quickly generate data to extract events from and test our methods.

To carry out the simulation, we built custom modules into the existing iCasa smart home simulator [5]. This simulation platform allows for simulating autonomic systems, handling internal communications, injection of new components at runtime, deletion or change of existing components. As such, we used a basic scenario consisting of a house of 4 rooms, a single user, and an outdoor zone. All four rooms of the house are equipped with a temperature controller system, monitoring and controlling heaters (fig. 2).



☐	Type	Name
☐	▦	window_3
☐	▦	window_4
☐	▭	controller_4
☐	▭	controller_3
☐	●	presence_4
☐	▦	window_1
☐	▦	window_2

Fig. 2. View of the simulator's web interface provided by iCasa. The four rooms are visible, with their equipment and the user.

Using this basis, we implemented a scenario spanning over 420 days, and comprising a daily cycle of outdoor weather (temperature and sunlight), as well as user's movements. All these daily changes create non-noticeable events, serving as a background noise for our experiments. To produce outstanding events, we randomly generated around twenty events, spanning over the whole duration of the simulation, of different kinds:

- Unusual weather: the outdoor conditions are set to unusually high or low temeperatures.
- Heater failures: heater can fail, making them turning off regardless of the command they receive.
- User's vacation: the user go out of the building for an extended period of time.
- Device removal/addition: a device is removed, or another one is added to the system.

The values of all devices and zones variables was regularly monitored throughout the simulation run, and the resulting data, which an excerpt is shown in figure 3 can later be used as a basis for our experiments.

B. *Implementing the complexity computation*

For the implementation of our method, we first needed to identify and characterize events from the time series data generated by the iCasa simulation. Since this is not the focus point of our present work (see sec. V), we simply apply threshold and precomputed conditions based detections to create a base of events.

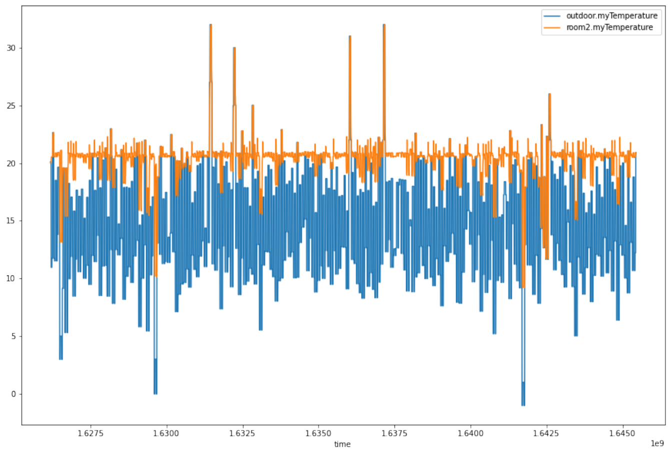


Fig. 3. Time series data from the simulation: outdoor temperature and controller temperature of a room. We can see two temperature drops, one for each room. What could be the causes of these drops?

This base of events consists the basis of the initial memory $\mathcal{M}$ used for computations. We then implemented a small number of predicate concepts:

- `label(e, k)`: whether the event e has the label ranked k^{th} in the memory.
- `rank(e, k)`: whether the event e has the rank k_1 along axis k_2, where k is a unique encoding for (k_1, k_2).
- `day(e, k)`: whether the event e occured k days ago.
- `month(e, k)`: whether the event e occured k months ago.
- `location(e, k)`: whether the event e occured in zone k.

With a straightforward implemention of memory, predicates and filters, the implementation of alg. 1 worked, but, as expected, took too long to be usable in realistic scenarios with hundreds or thousands of events to consider. In order to facilitate and speed up computaitons, we also implemented the following improvements:

- The memory object was augmented with various built-in rankings, allowing for faster operations during future filtering. For instance, since the memory object keeps a mapping from timestamps to events, which allows to quickly filter by date without having to loop over each stored element. This mapping, however, is not directly used to retrieve an event from the outside, as to preserve the theoretical model of memory as an unordered set presented in section III.
- Each of these predicates holds the property that, in addition to `True` and `False`, they can return another value, `None`, which is theoretically treated as `False` but carries the additional information that this predicate concept will also be false for any other element of the memory for any subsequent program k. This allows to effectively break the innermost loop in alg. 1.
- Some of the filters, for instance date or rank filters, were hard-written to select from the pre-computed mappings of the memory objects rather than testing a predicate over all memory elements.

C. Results

1) Memorability: The results of the descritpion complexity evaluation, for the described setup, are shown in fig. 4. The entire computation, over 4 iterations (meaning that retrieval paths contained at most 4 filters), took around 30 seconds on a modern commercial laptop with an i7-8700u CPU.

A main sequence of "usual" events, which complexity is roughly a logarithm of the elapsed time since their occurence, is visible. This corresponds to events for which the best retrieval path consists of a time description (e.g. "2 months and 12 days ago"). On the other hand, some events stand out in terms of complexity: some appear simpler, as they can be distinguished by using their rank alongside an axis ("the hottest day", "the second longest user's absence"), or the rare occurence of their kind ("the only fault on the heater").

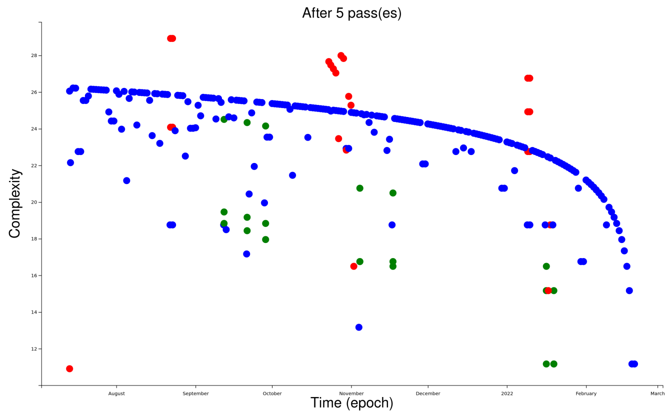


Fig. 4. The computed complexities of events with at most retrieval paths of length at most 4. Events of type "day" (blue), "hot" (green), "cold" (red) are shown.

Computing the "memorability score", which is shown in fig. 5, highlights these aforementioned events from the main "usual" sequence. Since the computation of this measure treats unusually complex or simple events the same way (from the absolute value operation in eq. 4), we observe that some events are memorabale due to their context only. For instance, a temperature anomaly occurring simultaneously to many other anomalies is notable, as it is costlier than expected to distinguish it from its neighbours.

Given that we generated the data used for this experiment, it is possible to flag all perturbation events from the usual daily events and evaluate how a detection based on "memorability" score would succeed in distinguishing these events. The result is presented as a ROC curve in fig. 6.

2) Abduction: As an illustration of the abductive inference possible by using the memorability score, we used as a targed consequence the temperature drop visible in the raw time series data from our scenario (fig 3).

TODO

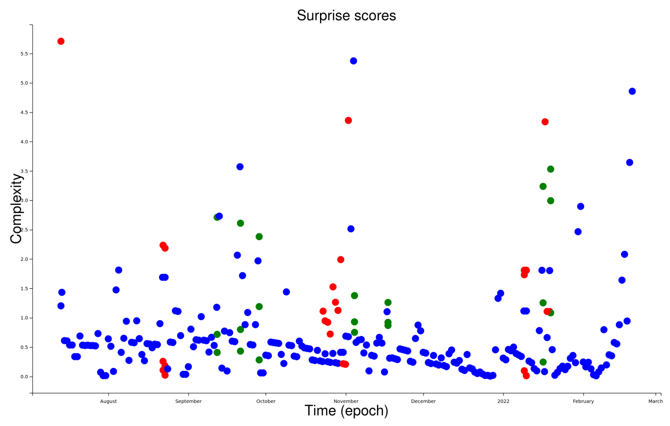


Fig. 5. Unexpectedness score for events in the memory

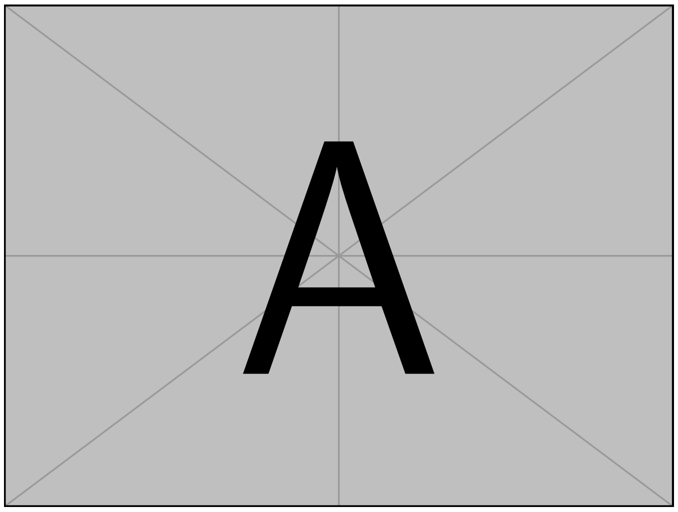


Fig. 6. Experimental ROC curve (True Positive Rate against False Positive Rate) for a classifier based on our memorability score.

V. Related Works

TODO

VI. Perspectives

The main purpose of the present paper is to show the possible connections between exising definitions of simplicity from cognitive science, Algorithmic Information Theory and a practical use case in cyber-physical systems. As such, many further improvements can be done to pave the way towards a better integration and performance for anomaly detection or abduction.

First, the main limitation of the current approach is the requirement of predefined predicate concepts, from which the different filters are constructed. As an extension, we suggest that in the future, we could explore online generation of such predicate. A posisbility would be to analyze discriminating dimensions of incoming data and create predicate as to name these differences, similar to the contrast operations proposed in [6], [7]. For instance, the predicate concept `hot` can be discovered by discriminating a recent hot day along the temperature axis and naming the difference with the prototypical day.

While the execution time is not part of the theoretical view of complexity, it is of prime importance for practical applications, especially when one considers implementation into real-time systems or embedded devices. While the computation we propose appear to be heavy, and possibly heavier as the number of allowed predicates grows, significant time savings can be achieve by trimming the base memory of past events deemed the most "non memorable". For instance, one can only retains the 100 most memorable events from the past. The difficulty with this approach is that such operations should be done in a manner to not interfere with the complexity computations for new elements: by forgetting some past events, even uninteresting ones, one should make sure to keep track of what made the interesting ones, interesting. Investigation of how to do so can pave the way towards practical implementations and dynamic selection of interesting events and help reducing the memory and computation cost of data-driven applications.

VII. Conclusion

References

[1] A. N. Kolmogorov, "Three approaches to the definition of the concept "quantity of information"," *Problemy peredachi informatsii*, vol. 1, no. 1, pp. 3–11, 1965, publisher: Russian Academy of Sciences, Branch of Informatics, Computer Equipment and . . . .

[2] M. Li, P. Vitányi, and others, *An introduction to Kolmogorov complexity and its applications*. Springer, 2008, vol. 3.

[3] P.-A. Murena, M. Al-Ghossein, J.-L. Dessalles, A. Cornuéjols, and others, "Solving Analogies on Words based on Minimal Complexity Transformations," in *International Joint Conference on Artificial Intelligence*. IJCAI, 2020.

[4] J.-L. Dessalles, "Coincidences and the encounter problem: A formal account," *arXiv preprint arXiv:1106.3932*, 2011.

[5] P. Lalanda, E. Gerber-Gaillard, and S. Chollet, "Self-Aware Context in Smart Home Pervasive Platforms," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*, 2017, pp. 119–124.

[6] J.-L. Dessalles, "From conceptual spaces to predicates," in *Applications of conceptual spaces: The case for geometric knowledge representation*, F. Zenker and P. Gärdenfors, Eds. Dordrecht: Springer, 2015, pp. 17–31. [Online]. Available: http://www.dessalles.fr/papers/Dessalles_14050201.pdf

[7] P. Gärdenfors, *Conceptual spaces: The geometry of thought*. MIT press, 2004.