

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 techniques. 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

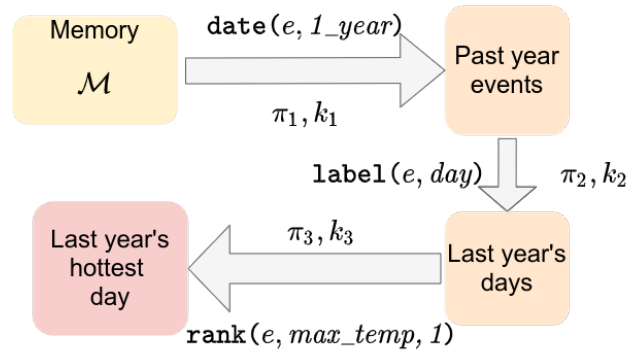
As the number of connected devices and sensors grows, so does the amount of produced data. As a consequence, the ability to filter out a few events from past recordings as “memorable” becomes prime. While this task is often overlooked by many statistical approaches, which instead benefit from large quantity of information, it is part of the learning process for humans [1]. Faced with new situations, humans are prone to infer previously seen unusual events as possible causes, and carry out further testing to assess the existence of a causal relation.

For instance, it is possible, for humans without any prior knowledge about physics or electrical engineering, to infer a memorable thunderstorm as a possible cause for a general black-out just by using the memorability of the former. While this kind of abductive inference (i.e. inference to the cause of an observation [2]) can yield many false-positives, it can be considered in situations where the lack of knowledge from previous occurrences penalizes other approaches.

This kind of abduction seems however hard to translate into machine, as the difficulty comes from different factors. First, events can be of different nature, and not directly comparable. In fact, even for specific systems such as smart homes, events range from device removal to a presence detection or an unusually high temperature. How can one then assess which ones are more memorable? Furthermore, even for comparable

events, different characteristics 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 use as a starting point the observation that while all events, regardless of their characteristics or nature, can be uniquely described using a combination of qualifiers, the most memorable ones are likely to require less words to be described. Think, for instance, of “last year’s hottest day” and “the 182th day of 7 years ago”. By evaluating the length of each description, taking into account both the complexity of concept words (a time ranking, a temperature ranking), and the arguments used in both descriptions (the hottest, the 182th, 7), we can assign each event a complexity score. Memorable events, as they stand out, would then differ from their neighbors by being much simpler (or more complex).



$$K(e_h) \leq l((\pi_1, k_1)) + l((\pi_2, k_2)) + l((\pi_3, k_3))$$

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 (gray arrows). The operations yields successive subsets of the memory (yellow), until a final singleton memory is reached, meaning that the successive filters retrieve a unique event from the memory. The predicates used for this retrieval 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”.

To programmatically achieve this computation, we rely on principles from Algorithmic Information Theory, which provides formal definitions of complexity and simplicity of objects, which appear to be consistent with human perception [3]–[5]. Figure 1 shows our solution applied to 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_k to describe them. These predicates thus provide filters $f_{\pi,k}$ that can be applied to the memory, selecting all events satisfying predicate π_k . By applying successive filters, we narrow down the size of the memory, until a single event remains. All the successive filters used to come to this singleton memory constitutes a *retrieval path*, whose length, in bits, gives an upper bound of the description complexity of the retrieved event.

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 unexpectedness 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 section VI

II. THEORETICAL BACKGROUND

Kolmogorov complexity formally quantifies the amount of information required for the computation of a finite binary string¹ (or any object represented by a finite binary string) [3], [6]. 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 .

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

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:

$$K_U(s) \leq l(p_u) + K_{U'}(s) \quad (2)$$

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 ,

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

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

¹While the definition still holds for some infinite 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 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 [3]. 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 .

Interestingly, Kolmogorov’s definition of complexity matches the intuitive notion and perception of complexity from a human standpoint. For instance, the complexity of short binary strings evaluated in [5] shows similar results to human perception of complex strings and patterns. More recently, [7] used Kolmogorov complexity to solve analogies and showed results close to human expectations.

This bridge between Algorithmic Information Theory and human perception of complexity is extended to define the notion of simplicity and unexpectedness, which are considered of uttermost importance in cognitive science [8]. [4] proposes a formal definition of the unexpectedness $U(e)$ of an event, as the difference between an a-priori expected causal complexity $C_w(e)$ and the actual observed complexity $C(e)$.

$$U(e) = |C_w(e) - C(e)| \quad (4)$$

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 simplicity or complexity of events.

Using this measure, it is possible to model phenomenons such as coincidences: imagine that you happen to run into someone in a park. If this person is not noticeable, the event will be quite trivial. On the other hand, this person is your best friend, as the description complexity $C(e)$ drops, the causal complexity $C_w(e)$ remains unchanged, and the event becomes noticeable.

Using these insights from AIT, we propose to measure the complexity of past events e from a memory \mathcal{M} by considering the quantity $K(e|\mathcal{M})$, that is the length of the shortest program outputting e , given the entire memory of past events \mathcal{M} as an input. To limit the search and obtain a computable measure, we limit ourselves to only a subset of possible programs, which we will now describe.

III. COMPUTING THE UNEXPECTEDNESS OF EVENTS

A. Retrieving an event

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:

$$e = (l, t, \mathcal{D}) \quad (5)$$

where l is the label, t a timestamp and \mathcal{D} a multi-dimensional data point 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:

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

An example of predicate is to use $\pi = \text{year}$ and $k = 1$, thus constructing the predicate $\text{year}(e, Y)$, 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.

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

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

As the output of a filter applied to the memory \mathcal{M} is another memory object $\mathcal{M}' = f_{\pi,k}(\mathcal{M})$, we can compose filter functions. A sequence of such filters, applied in succession, is called a *retrieval path*

$$p = (f_{\pi_1,k_1}, \dots, f_{\pi_n,k_n}) \quad (8)$$

and by definition $p(\mathcal{M}) = f_{\pi_n,k_n}(\dots(f_{\pi_1,k_1}(\mathcal{M})))$. In case the result of the operation $p(\mathcal{M})$ contains a single element e , we say that the path p *retrieves* the element e from \mathcal{M} , and write $p(\mathcal{M}) = e$.

B. Description complexity of events

As presented in sec. II, we are interested in computing the Kolmogorov complexity of an event e , conditioned on the memory \mathcal{M} . As this quantity is not computable, we use the previous definitions of retrieval paths and approximate the quantity $K(e|\mathcal{M})$ by the length of the shortest retrieval path p retrieving e from \mathcal{M} .

Given that all predicate concepts can be described by a prefix program of length $l(\pi)$, and their arguments in length $l(k)$, each retrieval path p can be described by the sequence of prefix description of all the predicate concepts and arguments used to construct its filters:

$$K(p) = l((f_{\pi_1,k_1}, \dots, f_{\pi_n,k_n})) = \sum_{i=0}^n l(\pi_i) + l(k_i) \quad (9)$$

Using this definition of the length of a retrieval path, we can compute the estimation for the complexity $K(e|\mathcal{M})$, which,

for clarity, we will denote simply by $K(e)$ for the rest of this paper.

$$K(e) \approx \min_{p|p(\mathcal{M})=e} K(p) = \min_{p|p(\mathcal{M})=e} \sum_{f_{\pi,k} \in p} l(\pi) + l(k) \quad (10)$$

Algorithm 1: Iterative computation of the approximate complexity

```

1 current_explore ← [(M, 0)] ;
2 future_explore ← [] ;
3 pass ← 0 ;
4 K(e) ← +∞ ;
5 while current_explore ≠ [] and pass < max_pass do
6   for (M_prev, K_prev) ∈ current_explore do
7     for π ∈ P do
8       for k ∈ {0, 1}* do
9         K_current ← l(π) + l(k) + K_prev ;
10        if K_current > max_complex then
11          break ;
12        end
13        M' ← f_{π,k}(M_prev) ;
14        if M' = {e} then
15          K(e) ← min(K(e), K_current) ;
16        else
17          future_explore.append((M', K_current));
18        end
19      end
20    end
21  end
22  current_explore ← future_explore ;
23  future_explore ← [] ;
24  pass ← pass + 1;
25 end

```

The definition used for description complexity in eq. 10 allows for a direct implementation, 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 from a given set \mathcal{P} 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 $(M', \text{len}(\pi, k))$ in an array `future_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 10.

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 behavior, 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 .

$$N_\pi(e) = \{e' \in \mathcal{M}, \pi(e') \wedge e' \neq e\} \quad (11)$$

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

$$K_{exp}(e) = \frac{\sum_\pi \sum_{e' \in N_\pi(e)} K(e')}{\sum_\pi |N_\pi(e)|} \quad (12)$$

The definition used for the expected complexity of the event e in eq. 12 holds the intuition of favoring more similar events: if e' is *very* similar to e , it will appear in many neighborhoods, since it satisfies mostly the same predicates as e . Therefore, its complexity term will be present in more terms in eq. 12, 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 consequence. 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. 10 and using relative filters.

$$Abd(c) = \min_{p|p(\mathcal{M}|e)=c} K(p) = \min_{p|p(\mathcal{M}|e)=c} \sum_{f_{\pi,k} \in p} l(\pi) + l(k) \quad (13)$$

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 possible 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 dependent 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 behavior 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 [9]. 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).

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 temperatures.
- 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.

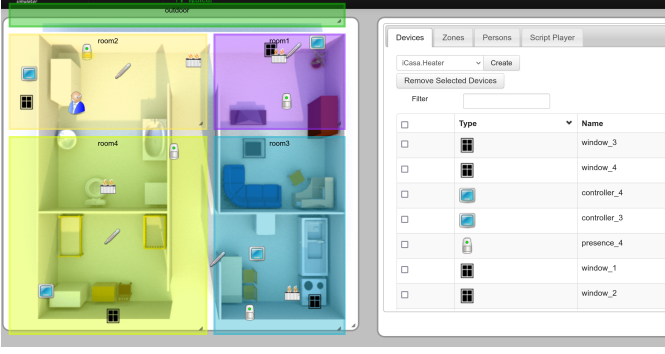


Fig. 2. View of the simulator's web interface provided by iCasa. The four rooms are visible, with their equipment and the user.

- Device removal/addition: a device is removed, or another one is added to the system.

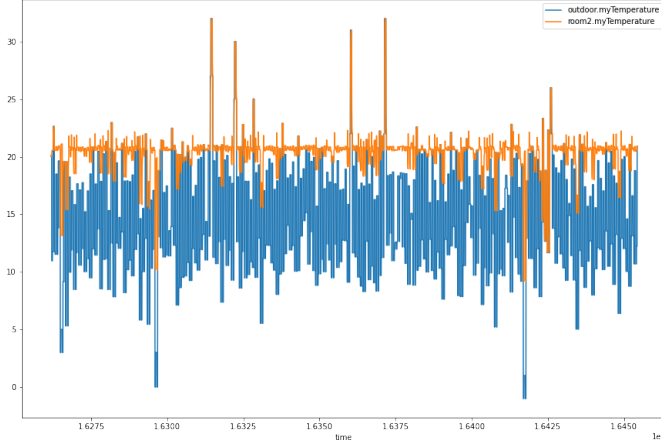


Fig. 3. Time series data from the simulation: outdoor temperature (blue) and controller temperature of a room (blue). On different occasion, the controlled temperature deviates from its setting (21°C). We will evaluate if our system finds these deviations as memorable, and if it can propose causal hypotheses.

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 pre-computed conditions based detection to create a base of events.

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:

- $\text{label}(e, k)$: whether the event e has the label ranked k^{th} in the memory.
- $\text{rank}(e, k)$: whether the event e has the rank k_1 along axis k_2 , where k is a unique encoding for (k_1, k_2) .

- $\text{day}(e, k)$: whether the event e occurred k days ago.
- $\text{month}(e, k)$: whether the event e occurred k months ago.
- $\text{location}(e, k)$: whether the event e occurred in zone k .

With a straightforward implementation 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 computations, 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 description 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 occurrence, 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 occurrence of their kind (“the only fault on the heater”).

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 memorable 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 neighbors.

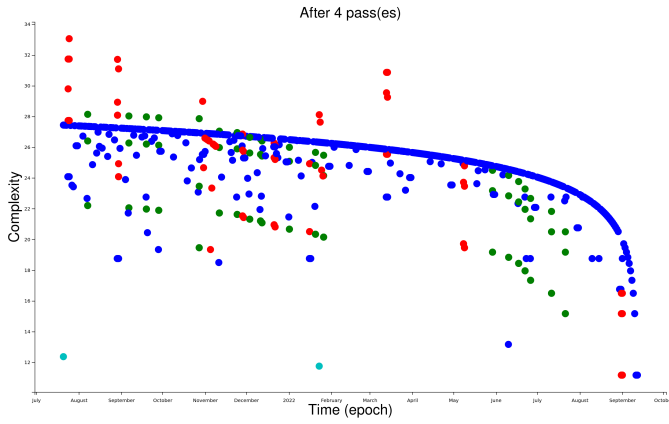


Fig. 4. The computed complexities of events with at most retrieval paths of length at most 4. Events of type “day” (blue), “hot” (green), “cold” (red), “device removal”(cyan) are shown.

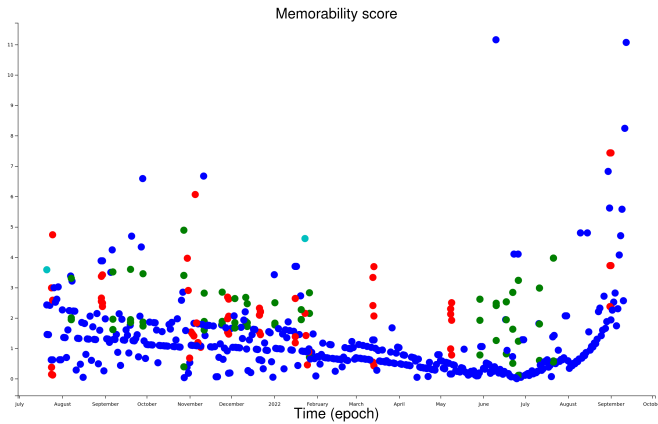


Fig. 5. Memorability score for events in the memory

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 target consequence the temperature drop visible in the raw time series data from our scenario (fig 3).

TODO

V. RELATED WORKS

TODO

VI. PERSPECTIVES

The main purpose of the present paper is to show the possible connections between existing 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.

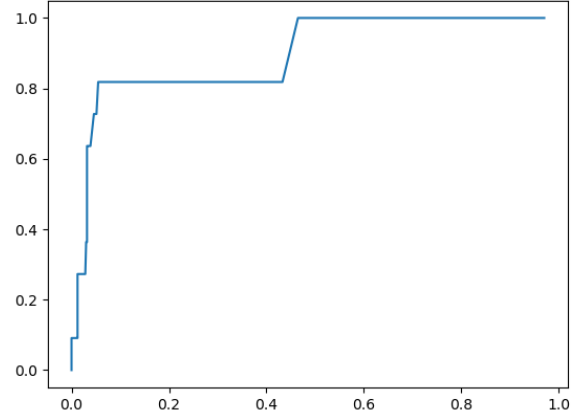


Fig. 6. Experimental ROC curve (True Positive Rate against False Positive Rate) for a classifier based on our memorability score.

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 possibility would be to analyze discriminating dimensions of incoming data and create predicate as to name these differences, similar to the contrast operations proposed in [10], [11]. 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 achieved by trimming the base memory of past events deemed the most “non memorable”. For instance, one can only retain 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] S. Dehaene, *How We Learn: The New Science of Education and the Brain*. Penguin UK, 2020.
- [2] L. Magnani, *Abduction, reason and science: Processes of discovery and explanation*. Springer Science & Business Media, 2011.
- [3] M. Li, P. Vitányi, and others, *An introduction to Kolmogorov complexity and its applications*. Springer, 2008, vol. 3.

- [4] J.-L. Dessalles, "Coincidences and the encounter problem: A formal account," *arXiv preprint arXiv:1106.3932*, 2011.
- [5] J.-P. Delahaye and H. Zenil, "Numerical evaluation of algorithmic complexity for short strings: A glance into the innermost structure of randomness," *Applied Mathematics and Computation*, vol. 219, no. 1, pp. 63–77, 2012.
- [6] 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
- [7] 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.
- [8] N. Chater and P. Vitányi, "Simplicity: A unifying principle in cognitive science?" *Trends in cognitive sciences*, vol. 7, no. 1, pp. 19–22, 2003, publisher: Elsevier.
- [9] 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.
- [10] 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
- [11] P. Gärdenfors, *Conceptual spaces: The geometry of thought*. MIT press, 2004.