Article

What should I notice? Using Alogithmic Information Theory to evaluate the memorability of events in smart homes

Étienne Houzé [1,2]*, Jean-Louis Dessalles [2], Ada Diaconescu[2] and David Menga [1]

[1] EDF R&D ; {first}.{last}@edf.fr; 7 boulevard Gaspard Monge, 91120 Palaiseau, France
[2] Télécom Paris; {first}.{last}@telecom-paris.fr; 19 place Marguerite Perey, 91120 Palaiseau, France
* Correspondence: etienne.houze@telecom-paris.fr

Abstract: With the increasing number of connected devices, complex systems such as smart homes record a multitude of events of various types, magnitude and characteristics. Faced with this variety and number, current systems struggle to identify which events can be considered more memorable than others. On the other hand, human beings are able, without knowledge of the system's inner working or large previous datasets, to quickly categorize some events as being more "memorable" than others. Having this ability would allow the system to identify and summarize a situation to the user or use the most memorable events as possible hypotheses in an abducitve inference process. Our proposal is to use Algorithmic Information Theory to formally define a "memorability" score based on the concept of retrieving events by using predicative filters. After providing a theoretical framework, we provide smart-home examples to illustrate how our approach could be implemented in practive and qualitatively compare our "memorability" score with human perception.

Keywords: Kolmogorov Complexity; Algorithmic Information Theory; Simplicity; Abduction; Surprise

Citation: Lastname, F.; Lastname, F.; Lastname, F. What should I notice?. *Entropy* **2021**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction

As a user has just switch on the TV in her all-equipped living-room, the lights dim and the window blinds go down. Intrigued by this behavior, she quickly infers that both light dimming and blind closing occurred as a consequence of the TV being turned on. How did she come to this conclusion? By performing *abductive inference* [1]. This mental operation is a key element of humans' ability to understand the world: from the observed consequences, they infer the possible causes.

In this example, there are mainly three possibilities to come to the conclusion. (1) If the user knows how the smart living-room system works, if she knows the underlying rules or parameters, she may use this causal knowledge to perform abduction. (2) If she has no knowledge about the system but made several observation of the same behavior, she may examine past correlations and figure out that turning on the TV set often leads the blinds to close and the lights to dim. (3) If there are no previous occurrences of the event (e.g. it is the first time she turns on the TV in the living-room), she may still be able to suspect that the TV is a possible cause for the observed event, just because it appears to her as a memorable recent event (as it is its first occurrence). This example suggests that human beings are able to use distinct methods to perform abductive tasks and infer new knowledge, depending on the situation. While the first two mechanisms can be automated using knowledge bases and statistical methods,the third approach, which can be used without any knowledge of the occurring phenomenon or past occurrences. Instead, it relies on the observation that some events appear more "memorable" than others and, as such, can be considered as possible hypotheses if need be.[1]

Defining a score of memorability is not straightforward. First, events can be of different nature, and not directly comparable. Even for restricted systems such as smart homes, noticeable events range from device removal to presence detection or unusually high temperatures. Even for comparable events, the problem is to weigh different characteristics: is a record-high temperature 47 days ago more memorable than the small deviation recorded just 3 minutes ago? To our knowledge, no current system proposes to combine various event types from different devices to compute a unified metrics of "memorability". In addition to the aforementionned use for abductive inference, having access to a computation of memorability would allow a system to summarize a situation to its user by presenting only the most memorable events: for instance, a summary of notable events that occurred during the home owner's absence.

To address this issue, we started from the following supposition: while all events, regardless of their characteristics or nature, can be uniquely described using a combination of quantitative or qualitative qualifiers, the most memorable ones are likely to require less words to be described. This supposition appears to be in line with some observations: for instance, a corrolation has been found between word frequency and length[2]. Thus, "last year's hottest day" is simpler than "the 182$^{\text{nd}}$ day 7 years ago". How can we quantify this relative simplicity? We could evaluate the complexity of each description, taking into account both the complexity of the concept words (a date of occurrence, a temperature ranking), and of the arguments (1st hottest, 182 and 7). The resulting values define the *description complexity* of events. Our intuition is that memorable events require simpler and less numerous qualifiers to be unambiguously described than unremarkable ones.

For machines to implement and compute description complexity, we need a formal framework and computation methods that incorporate the aforementionned process. Algorithmic Information Theory (AIT) appears to be such a framework, as it is consistent with the human perception of complexity[3–5]. Our method is as follows: we consider events as being elements stored in what we call *base memory*. To reproduce the language features applicable to events, we use *predicates*, i.e. functions assigning a boolean value to events. For instance, the predicate $\texttt{date}(\cdot, 1\_year)$ is *true* of events that occurred last year. Selecting all events, from the memory, that satisfy a given predicate corresponds

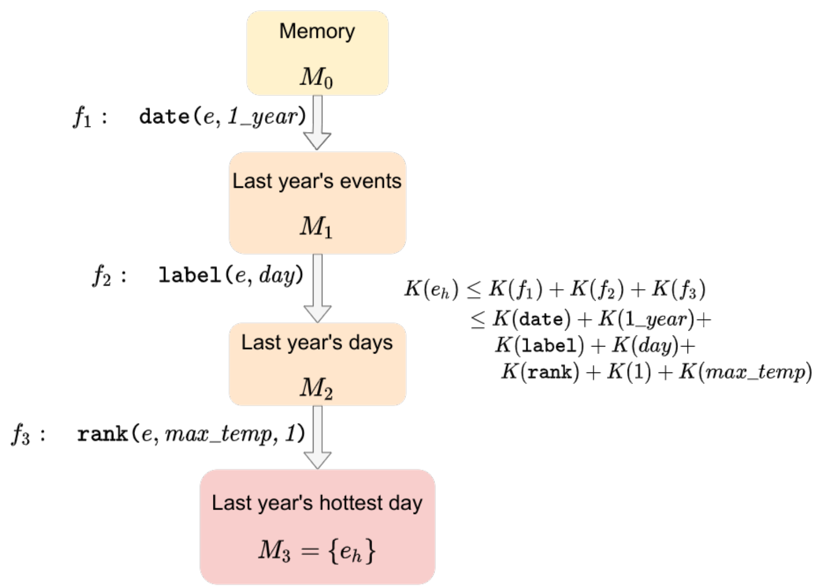


Figure 1. Retrieving an event through successive predicative filters. From the base memory (yellow), successive filters select events satisfying the associated predicate (gray arrows). For example, filter f_1 selects events from last year, i.e. which satisfy the predicate $\texttt{date}(\mathit{event}, 1\_year)$. In this case, successively applying filters f_1, f_2 and f_3 yields a unique event e_h, last year's hottest day. The complexity of this event can then be upper-bounded by the complexity of the three filters as they give an unambiguous way to describe the event within the memory.

to a *filter* operation. It generates another memory that is a subset of the previous one. The filtering operation can then be repeated, selecting fewer events at each iteration, until a singleton memory is reached. This means that the sequence of predicates could unambiguously *retrieve* the unique remaining event. The description complexity of this event can thus be upper-bounded by the number of bits required to describe the filters used in the retrieval process. Figure 1 illustrates this process for the event: "last year's hottest day".

The rest of this article is organized as follows. We will first briefly introduce some basic notions of Algorithmic Information Theory in subsection 2.1. We then expose our contribution by presenting in subsection 2.2 our formal definition of memorability and related notions. Then, we present an implementation example of these definitions with two smart-home examples in section 3. The results of these experiments are then presented and discussed. Finally we explore other related works in subsection 4 and explore possible extensions of our work in subsection 5.

2. Theoretical Framework

2.1. Background

Kolmogorov complexity formally quantifies the amount of information required for the computation of a finite binary string[1] (or any object represented by a finite binary string)[3,6]. The complexity $K(s)$ of a (finite) binary string is the length in bits $L(p)$ 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 Equation 1, all results hold, up to an additional constant, if we change the machine. Think how any Turing-complete programming language can be turned into any other language, using an interpreter or a compiler program. 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 complexity $K(s)$, based on the choice of a reference Turing machine, such that, for any other machine U taken from the set TM of Turing machines:

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

where the additional constant C_U does not depend on s.

Note that the notion of Kolmogorov complexity involves no requirement on the execution time of programs, only their length in bits matters for the computation of complexity. Though Kolmogorov complexity can be shown to be incomputable[3], it can be approximated with upper bounds by exhibiting a program outputting s.

Interestingly, Kolmogorov 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.

The bridge between Algorithmic Information Theory (AIT) and human perception of complexity can be pushed farther thanks to the notions of simplicity and unexpectedness, which are sometimes regarded of uttermost importance in cognitive science[8]. [4]

[1] Though the definition holds for some infinite binary strings (think of the representation of the decimals of π), we restrict ourselves here to finite strings.

proposes a formal definition of the unexpectedness $U(e)$ of an event, as the difference between an a-priori expected causal complexity $K_w(e)$ and the actual observed complexity $K(e)$.

$$Unex(e) = K_w(e) - K(e) \tag{4}$$

This result comes from the understanding that, while Kolmogorov complexity is ideally computed using a Turing machine, it can be used as a proxy for modeling information processing in the human brain, and thus can be used to define a notion of simplicity or complexity of events.

Definition 4 allows to model phenomena such as coincidences: imagine that you happen to run into someone in a park. If this person has no particular link to you, the event will be quite trivial: the complexity of describing this person will be equivalent to distinguishing it from the global population, which is also roughly equivalent to the (causal) complexity of describing the circumstances having brought this person to be in that park as the same time as you. On the other hand, if you run into your best friend in a park, as the complexity of describing your best friend is significantly lower, the description complexity $K(e)$ drops while the causal complexity $K_W(e)$ remains unchanged. This is why this latter event appears unexpected.

As [4] suggests a link between the unexpectedness and cognitive relevance, we propose to define the memorability of an event in a similar way. Since we want to use this score in applications, we need a definition that is well-defined and computable in practice. Hence, we define the memorability $M(e)$ of an event as the absolute difference between its description complexity $K_d(e)$ and its expected description complexity $K_{exp}(e)$:

$$M(e) = |K_{exp}(e) - K_d(e)| \tag{5}$$

Contrary to the definition of unexpectedness from Equation 4, we use an absolute value: we do so to acknowledge the fact that events more complex than expected can be memorable as well[2]. In the next section, we define computational approximations for the description complexity K_d and the expected complexity K_{exp} of events.

2.2. Computing the memorability of events

2.2.1. Retrieving an event

We define *events* as data points augmented with a *label* indicating their nature (temperature event, failure event, addition/removal of a device) and a timestamp of occurrence. Formally:

$$e = (l, t, \mathcal{D}) \tag{6}$$

where l is the label, t the timestamp and $\mathcal{D}$ a vector of properties characterizing e: its duration, the maximum temperature reached, the sensor name, its position, etc. Labels can also be considered as classes of events, of which each event is a particular instance.

To model how humans are able to describe events by using qualifiers, we use *predicates*: Boolean functions operating on events and, possibly, additional parameters: $\pi(e, a_1, a_2, \dots, a_n) \mapsto \{O, 1\}$ is a predicate of arity n operating on event e. In the rest of this paper, we will prefer the equivalent notation $\pi(e, k) \mapsto \{0, 1\}$, where k is a binary string encoding the sequence of arguments $a_1, \dots, a_n$. Using this notation, the predicate π becomes a boolean function operating on $\mathbf{E} \times \{0, 1\}^*$, where $\mathbf{E}$ denotes the set of all events:

$$\pi : \begin{cases} \mathbf{E} \times \{0,1\}^* & \mapsto \{0,1\} \\ (e,k) & \mapsto \pi_k(e) \end{cases} \tag{7}$$

[2] In the original paper [4], exceptionally complex events are described by considering complexity itself as a way to describe the event: see "the Pisa Tower effect"[9]

As an example of predicate, consider $\pi = \texttt{year}$ and k a string encoding the number 1, thus constructing the predicate $\texttt{year}(e, 1)$, which tells whether the event e occurred 1 year ago.

As events occur, they are stored in the *base memory* M_0. As they are not directly comparable, the memory M_0 can be considered as having the structure of an unordered set. We denote by $\mathcal{M}$ the set of all subsets of M_0. By extension, elements of $\mathcal{M}$, i.e. subsets of M_0, are also called *memories*.

By applying a given predicate π to all events contained in a memory $M \subseteq M_0$, and selecting only events satisfying π, one gets another memory $M_1 \subseteq M \subseteq M_0$. We call this operation a *filter*:

$$f_{\pi,k} : \begin{cases} \mathcal{M} & \mapsto \mathcal{M} \\ M & \mapsto \{e \in M | \pi_k(e)\} \end{cases} \tag{8}$$

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

As the output of a filter applied to a memory M is another memory object $M' \subseteq M$, we can compose filter functions. A sequence of such filters is called a *retrieval path*

$$p = (f_{\pi_1,k_1}, \ldots, f_{\pi_n,k_n}) \tag{9}$$

By definition $p(M) = f_{\pi_n,k_n}(\ldots(f_{\pi_1,k_1}(M)))$. In case the result of the operation $p(M)$ contains a single element e, we say that the path p *retrieves* the element e from M, and write $p(M) = e$. In the example shown in Figure 1, the three filters f_1, f_2, f_3 form a retrieval path retrieving the event "last year's hottest day" from the base memory M_0.

2.2.2. Description complexity of events

As presented in sec. 2.1, we are interested in computing an approximation of the description complexity of an event e. From the above definitions, if there is a path p retrieving e from the base memory M_0, i.e. $p(M_0) = e$, this path provides a possible unambiguous description for e. We therefore define the description complexity of e as the minimum complexity of a path p retrieving e from the base memory M_0.

$$K_d(e) = \min_{p \in P_\infty} \{L(p) | p(M_0) = e\} \tag{10}$$

where the bit-length $L(p)$ of a retrieval path is defined as the number of bits of a string encoding the path. If we limit ourselves to prefix-free strings encoding predicates and arguments, the total bit length is given by:

$$\begin{aligned} L(p) &= L((f_{\pi_1,k_1}, \ldots, f_{\pi_n,k_n})) && (11) \\ &= L(\pi_1) + L(k_1) + \cdots + L(\pi_n) + L(k_n) && (12) \end{aligned}$$

where $L(\pi_\square)$ and $L(k_\square)$ respectively denote the length, in bits, required to express the predicate's concept and program. This length may vary depending of the encoding choice, see Section 3 for an example.

By considering only a finite number of possible predicates π and arguments k, and a maximum path length, we can construct a finite set P of possible retrieval paths. By limiting the search over this set, we get an upper bound of description complexity, and use this upper bound as an approximation:

$$K_d(e) \leq \min_{p \in P \wedge p(M_0) = e} L(p) = \min_{p \in P \wedge p(M_0) = e} \sum_{f_{\pi,k} \in p} L(\pi) + L(k) \tag{13}$$

The approximation of description complexity from Equation 13 allows for a direct implementation, which is shown in Algorithm 1. This algorithm operates iteratively:

```
current_explore ← [(M,0)] ;
future_explore ←[ ] ;
pass ← 0 ;
K(e) ← +∞ ;
while current_explore ≠ [ ] and pass < max_pass do
    for (M_prev, K_prev) ∈ current_explore do
        for ß ∈ P do
            for k ∈ {0,1}* do
                K_current ← l(ß) + l(k) + K_prev ;
                if K_current > max_complex then
                    break ;
                end
                M' ← f_π,k(M_prev) ;
                if M' = {e} then
                    K(e) ← min(K(e), K_current) ;
                else
                    future_explore.append((M', K_current));
                end
            end
        end
    end
    current_explore ← future_explore ;
    future_explore ← [ ] ;
    pass ← pass + 1;
end
```

Algorithm 1: Iterative computation of the approximate complexity

starting from the base memory M_0 (line 1), we apply all possible predicate concepts π from a given finite set 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).

2.2.3. Computing Memorability

As stated in Equation 5, we define memorability $M(e)$ as the absolute difference between the description complexity of an event and its expected value. As we've just defined $K_d(e)$ and provided an approximation in Equation 13, we now focus on defining the *expected* description complexity of an event, $K_{exp}(e)$ that appears in Equation 5

$K_{exp}(e)$ evaluates the complexity that the user, or the system, would expect for the occurrence of event e to have, based on their previous knowledge. In our framework, this prior knowledge consists of the base memory M_0. The expected complexity of the event e can be computed with a simple first-order approximation, i.e. estimating the average complexity of "similar events" over the base memory M_0.

Still, there is a difficulty in defining what should be considered *similar* events. Given that we deal with non comparable events, we may define the notion of similarity by referring once again to *predicates*. For a given event e and a given predicate π_k, we define a π_k-neighborhood of e as the set $N_{\pi,k}(e)$ of all other events satisfying π_k.

$$N_{\pi,k}(e) = \{e' \in M_0, (e' \neq e) \wedge \pi_k(e')\} \tag{14}$$

Now, when considering, for all possible predicates π_k, the corresponding neighborhoods $N_{\pi,k}(e)$, with the convention that $N_{\pi,k}(e) = \emptyset$ if e does not satisfy π_k, we can compute an average expected complexity for e, by summing the complexity of events in all neighbourhoods of e and dividing the total by the number of events in the neighborhoods:

$$K_{exp}(e) = \frac{\sum_{\pi,k} \sum_{e' \in N_{\pi,k}(e)} K_d(e')}{\sum_{\pi,k} |N_{\pi,k}(e)|} \tag{15}$$

This definition is consistent with the intuitive idea that more similar events should weigh more in the computation. Indeed, if e' is very similar to e, it will appear in many neighborhoods, since it satisfies most of the predicates that e satisfies. Therefore, it will be present in more terms in Equation 15, and will weigh more in the final result.

2.2.4. From memorability to abduction

Abductive inference builds upon the computation of the memorability score. *Knowing* that we want to find a cause c for an observed effect e, we try to find the most remarkable event in past memory that is related to e. While our "memorability" score identifies remarkable past events, it does not take into account their relatedness to e.

The knowledge attached to the occurrence of e can be integrated into the description complexity definition by using conditional complexity $K_d(c|e)$: The information contained in e is considered as given, and therefore "free" in terms of complexity. For instance, when looking for a cause of an anomaly in the living-room, other anomalies occurring in the same living-room will be simpler, as the location "living-room" is already known from the observation of the current anomaly.

Formally, we now consider that knowledge of the effect is given. This consists, for instance, of appending a description of effect e is appended to all programs k: $\pi_{k::e}(c)$, where :: is the append operation. The set of paths obtained with such predicates is noted P_e^∞. This append operation is free in terms of bit-length in the computation of complexity, since the effect event e is an input of the problem. Therefore, we have $L'(\pi_{k::e}) = L(\pi_k) = L(\pi) + L(k)$. We get a definition for the conditional description complexity:

$$K_d(c|e) = \min_{p \in P_e^\infty} \{L'(p), \quad p(M_0) = c\} \tag{16}$$

$$= \min_{p \in P_e^\infty} \left\{ \sum_{f_{\pi,k::e} \in p} L(\pi) + L(k), \quad p(M_0) = c \right\} \tag{17}$$

This new conditional description complexity translates the additional information provided to the system when answering a user's request. It can then be averaged over similar events to compute the expected conditional description complexity, $K_{exp}(c|e)$. From this, we come to the definition of the conditional memorability, which measures how memorable an event c turns out to be in the context of the occurrence of another event e:

$$M(c|e) = |K_{exp}(c|e) - K_d(c|e)| \tag{18}$$

Conditional memorability encapsulates the idea presented as the motivation of this paper: when confronted with a surprising situation, and in the absence of any other source of information, events that appear more memorable than others, with regards to the target event will be selected as potential causes. As such, our conditional memorability score provides a ranking that can be used for abductive inference.

This metrics answers the different problems exposed in the introduction: by using a universal measure for complexity, bits, it allows to compare values from different dimensions. For instance, it solves the dilemma of recent events: is a big event a long time ago more memorable than a smaller one only a few minutes ago? The answer from conditional memorability is that "it depends". It depends on the complexity cost set for the predicates describing time and the magnitude of the event. These complexity costs should reflect the user's perception of the different dimensions (see Section 5).

3. Experiments

3.1. Setup

We design two different setups to test our approach. Both are inspired from smart home use cases. 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 experience atypical situations, highly dependent on the context, for which pre-established relations might fail to find good abduction candidates. Our choice was also motivated by the existence of previous work[10] involving smart home simulations capable of quickly generating data from which we could extract events and test our methods.

3.1.1. Scenario 1

In this setup, we aim to reproduce the example mentioned in Section 1: the installation of a brand new smart TV causes unpredictable effects on the light of a room. To play this situation, we created a set of events covering a period of 100 days, corresponding to the past knowledge of the house. Two kinds of events are recorded: "TV event", corresponding to TV use, and luminosity events, describing the luminosity of the room at a given time. Low lights occur at night, and can occasionally occur during daytime, with a small probability. On the 100$^{\text{th}}$ day, a "TV event" is recorded with a different "device" characteristic. Shortly after, the light lowers, which is recorded in the following "light" event.

3.1.2. Scenario 2

We consider an experimental smart home setup, with various sensors, which we simulate over a period of time. To carry out the simulation, we built custom modules into the existing iCasa smart home simulator[10]. ICasa offers a simulation of autonomic systems that can handle internal communications, the possible insertion of new components at runtime, or the deletion or modification of existing components. We used a basic scenario consisting of a house with four rooms, a single user, and an outdoor zone. All four rooms are equipped with a temperature control system that controlling heaters (see Figure. 2).

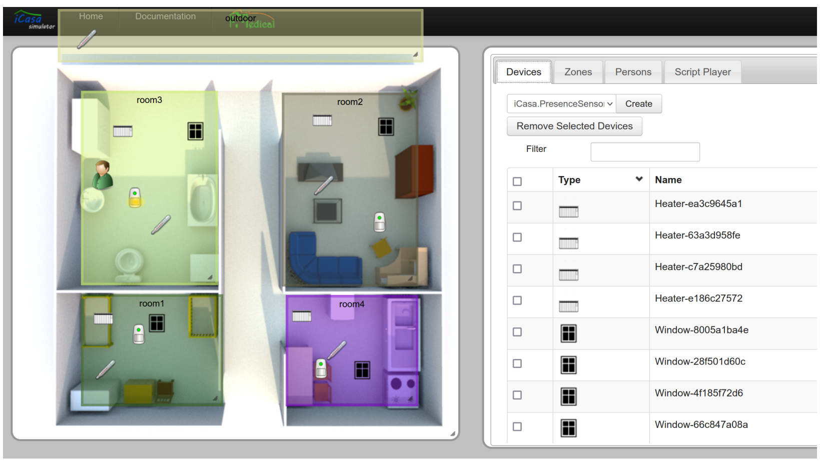


Figure 2. View of the simulator's web interface provided by iCasa. The four rooms are visible, with their equipment and the user.

Based on this, we implemented a scenario spanning over 420 days, and comprising a daily cycle of outdoor weather (temperature and sunlight) fluctuations, as well as user's movements. All these daily changes create non-noticeable events, serving as a background for our experiments. To produce outstanding events, we randomly generated about 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 may break down, making them turn off regardless of the command they receive.
- Device removal/addition: a device is removed, or another one is added to the system.

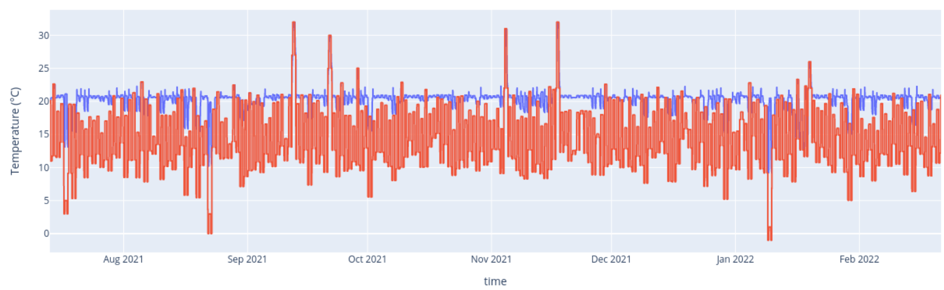


Figure 3. Time series data from the simulation: outdoor temperature (red) and controller temperature of a room (blue). To be used in our framework, these time series data are processed by a simple threshold-based event detection.

Values observed from all devices and zones were sampled throughout the simulation. The resulting data (figure 3) was then used as a basis for our experiments. We then process the time series data to identify and characterize events. Since how events are detected is not the focus point of our present work (see Section. 4), we perform event detection merely based on threshold and precomputed conditions.

3.2. Implementing the complexity computation

We implemented the computation of both the description complexity and the memorability score into a Python object, called the `SurpriseAbductionModule`. Apart from the base memory of events M_0, this module contains a set of predefined predicates Π to characterize events. For instance, for scenario 2, the predicates we use are the following:

- `label(e,k)`: whether the event e has the k^{th} most frequent label (meaning that frequent labels are simpler to express than rare ones)
- `rank(e,r,a)`: whether the event e is ranked r^{th} for characteristic a, where characteristics are encoded by their frequency (again, common characteristics are the simplest ones)
- `day(e,k)`: whether the event e occurred k days ago.
- `month(e,k)`: whether the event e occured k months ago.
- `location(e,k)`: whether the event e occurred in zone k.

The description length $L(\pi, k)$ of a predicate π_k is computed as follows: since the set of predicates is finite and known, $L(\pi) = \log 2(|\Pi|)$ are enough to describe the predicate concept π[3]. To encode the argument k of the predicate, we used the widely used prefix-

[3] This approach gives an equal complexity to all predicate concepts. While this could be argued when using many concepts, as humans do, we deemed better, for our small examples, to consider all concepts as equal.

free Elias delta code[11], which requires $L(k) = \log 2(k) + 2\log 2(\log 2(k) + 1) + 1]$ bits. The total cost of describing π_k therefore is

$$L(\pi, k) = \log 2(|\Pi|) + \log 2(k) + 2\log 2(\log 2(k) + 1) + 1 \quad (19)$$

With a straightforward implementation of memory, predicates and filters, we could run alg. 1 worked, though, as expected, it 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 timestamp to events one can perform a quick filtering by date without having to loop over all stored element. This convenient mapping, however, is not directly used to retrieve events by their date of occurrence, so as to preserve the theoretical model of memory as an unordered set, as presented in section 2.2.
- 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 the date and rank filters, were hard-written. Events can be selected from these precomputed mappings over the memory objects rather than by testing a predicate over all memory elements.

Our code is written in Python. Examples are presented in the form of Jupyter Notebooks, which allow to quickly reproduce our results and figures. All code is available on our Github: https://github.com/EtienneHouze/memorability_code.

3.3. Results

3.3.1. The "TV" scenario:

Memorability

The computation of the description complexity measure for the 2400 events recorded in this scenario took around 90 seconds using an i7-8600u laptop. The resulting complexities, in Figure 4, show that TV events appear simpler than luminosity events. In fact, for most luminosity events, the complexity corresponds to the random choice complexity, 25 bits in this example. This is due to the frequency of luminosity events: given that there are 24 of them per day, considering the retrieval path $day(k), randomPick(24)$ costs more bits than $randomPick(N)$, where N is the total number of events, with the notable exception of recent days, for which the value k is low (see the right-hand side of Figure 4).

Abduction

To show the potential application to abductive inference, we use our approach to the first scenario, to see if memorability alone can find the brand new TV to be a reasonable cause for the sudden low light.

The result of our algorithm is given in Table 1. It shows that the system correctly identifies the TV as being the cause. The reason for this choice is that, since the smart TV's device ID is unique among all other events of type "TV", its description complexity is very low.

3.3.2. The "temperature" scenario:

The results of the description complexity evaluation, for the described setup, are shown in Figure 5. The entire computation, over 4 iterations (meaning that retrieval

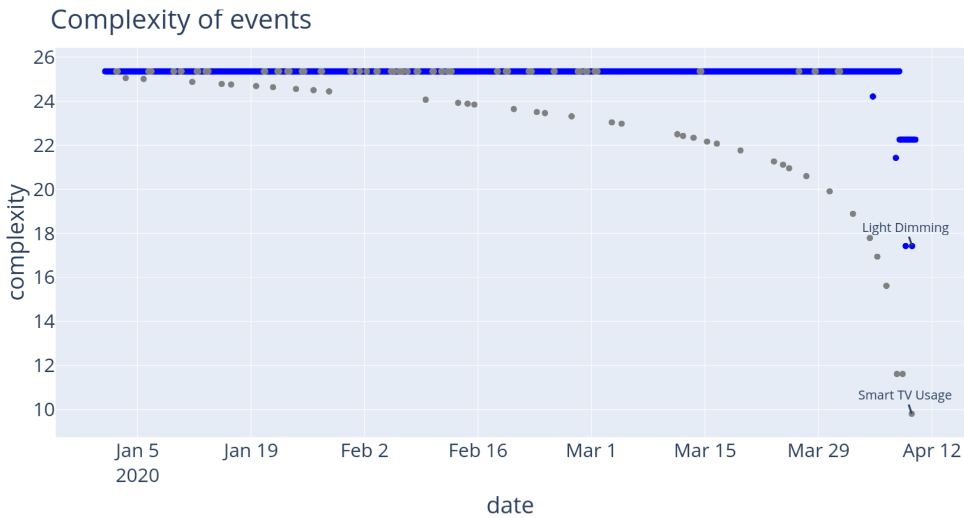


Figure 4. Description complexity for events recorded in the TV scenario. Luminosity events are shown in blue, TV events in green

Event Id	Description	Relative memorability (bits)
2513	Use of smart TV	16.76
2427	Last use of the old TV	14.81
2411	Second-last use of the old TV	11.21

Table 1: Output of the memorability-based abduction module: top 3 events for the relative memorability metrics.

paths contained at most 4 filters), took around 30 seconds on a commercial laptop with an i7-8700u CPU.

The general trend appears different from the "TV example" presented in Figure 4 : instead of having a capped complexity score, a logarithmic trend appears. This phenomenon is due to the absence, in this example, of the `randomPick` predicate; instead, usual events, with outstanding particularity, can be retrieved by using the `day` predicate and a `label` predicate. As such, it appears that most days are considered usual by our memorability score, creating the main logarithmic sequence of blue dots: there is no better way to retrieve them than simply giving the day of their occurrence. On the other hand, some events stand out in terms of complexity: some appear simpler, as they can be distinguished by using their rank along some axis ("the hottest day", "the second coldest day"), or the rare occurrence of their kind ("the only fault on the heater").

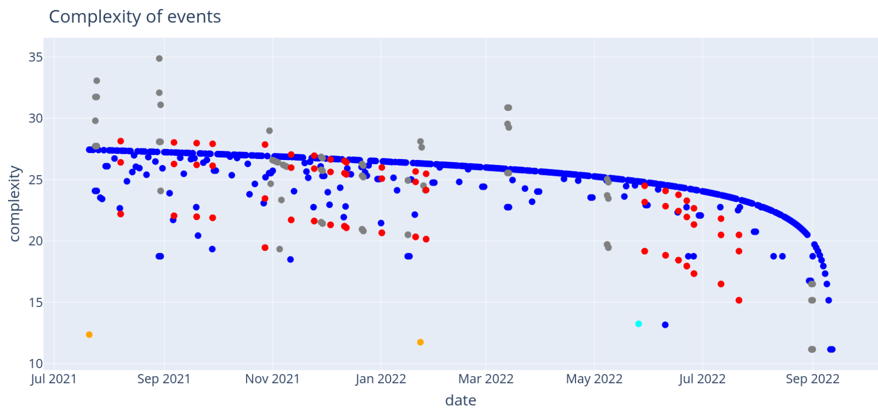


Figure 5. Computed description complexities of events with retrieval paths of length at most 4. Events of type "day" (blue), "hot" (red), "cold" (gray), "device removal"(orange) and "device addition" (cyan) are shown. Events mentioned in the text are specified.

Event iD	Event Type	Retrieval Path
20	day	Label("day"), AxisRank(0, "max_temp")
329	day	Label("day"), AxisRank(0, "min_temp")
183	deviceRemoval	Label("deviceRemoval"), Day(0)
149	cold	Label("cold"), Day(2), Device("thermo_2")

Table 2: Selected events from the “temperature example” with their shortest retrieval path.

The “memorability score” is shown in Figure 6. Similar to what happened with the description complexity score, most events appear with a low memorability: this corresponds mostly to events from the “main sequence” from Figure. 5. On the other hand, some events stand out : for instance events 20 and 329, which are respectively the hottest and coldest days recorded, or event 183 which correspond to the rare type *device_removal*. Since our memorability measure treats unusually complex or unusually simple events the same way (from the absolute value operation in Equation 4), we observe that some events are memorable due to their context only. For instance, the group to which event 149 belongs appears more complex than expected: the same event occurring simultaneously in all four rooms of the house make each instance harder to discern. Table 2 illustrates this by exhibiting the retrieval paths used for complexity computation for these events.

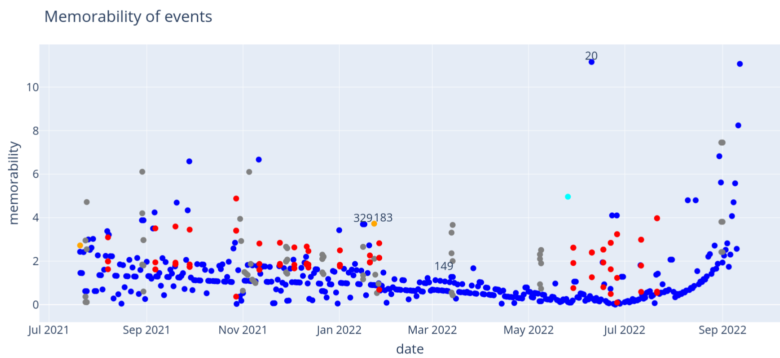


Figure 6. Memorability score for events in memory

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 perform in distinguishing these events. The result is presented as a ROC curve in Figure 7. This shows the memorability score’s performance as a classifying tool for “out-of-the norm” events. In this example, memorability alone correctly identified 18 manually generated events with only 5 false-positives. While the direct application of memorability for classification or event detection is not within the scope of this paper (see Section 4 for complexity-based detection), this first result is on par with the motivation of memorability being in accordance with the intuitive notion.

3.4. Discussion: the subjectivity of predicates

For individuals, the notion of memorability and complexity of events is highly subjective: the same event may appear usual for a person, while being exceptional for her neighbor. Our approach to memorbaility embraces this subjectivity, incorporating it through the definiton of predicates.

In Figure 8, we show this subjectivity by comparing the analyses of the same base memory of events, generated using the same setup as for Example 1, using different three different sets of predicates. The resulting figures show different phenomena based on the predicates at hand by the module. In figure 8a, only time and labels were recognized by predicates. As a result, the general trend of the curve is logarithmic, as events from T

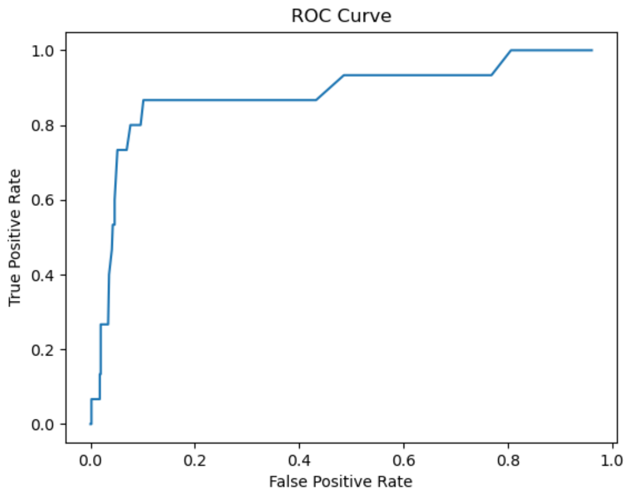
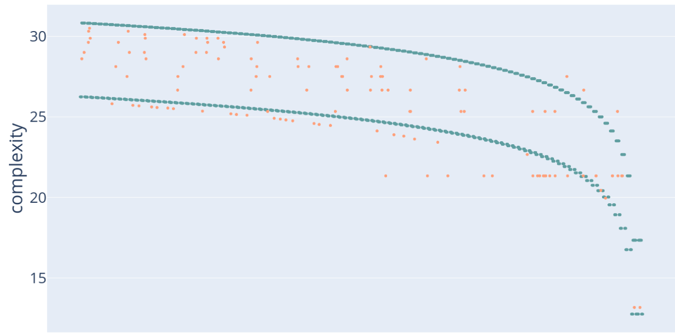


Figure 7. Experimental ROC curve (True Positive Rate against False Positive Rate) for a classifier based on our memorability score. Measures consider 23 manually flagged events as memorable (events added to the normal background as described in Section 3.3.)

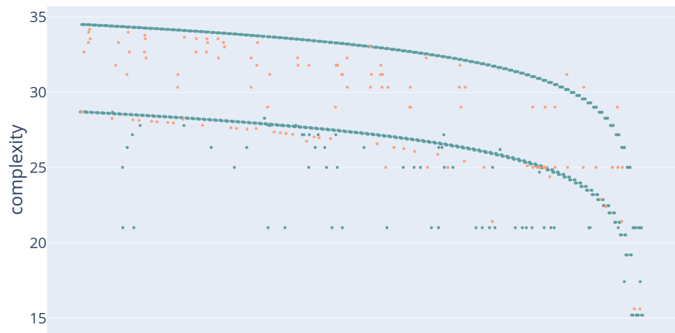
days ago require $O(\log(T)))$ bits of information. Two main sequences of light events appear: as some light events were recorded simultaneously as TV events, they require the additional information of their label to be retreived. In Figure 8b, we added predicates qualifying the light intensity, along with a day/night distinction. This added capacity isolated some light events as much simpler: they appear as a green horizontal line. These events are times when the light was low during day, or high during night. As such, they are outliers, and therefore require less inforamtion to be retrieved.

Finally, Figure 8c comes from a module which has the ability to retrieve any event from a memory of size N, at the expense of $O(\log(N))$ bits of information. This added capacity has the immediate effect of setting a clear upper limit to the description complexity of items, since any item can be retrieved using $\l(N)$ bits (in this example, this limit is around 25 bits). While this kind of direct retrieval is trivial in computer science (one could use the memory address of any stored event), its correlation with human cognition is not obvious: can humans be considered to have this possibility of selecting any event from their entire memory, without restriction regarding their nature, their age, their magnitude? However, this highlights an interesting phenomenon: using this direct retrieval, the module makes no distinction, complexity-wise, between events past a given threshold. In other words, there is a generic "uninteresting" category of events, in which the module makes no distinction.

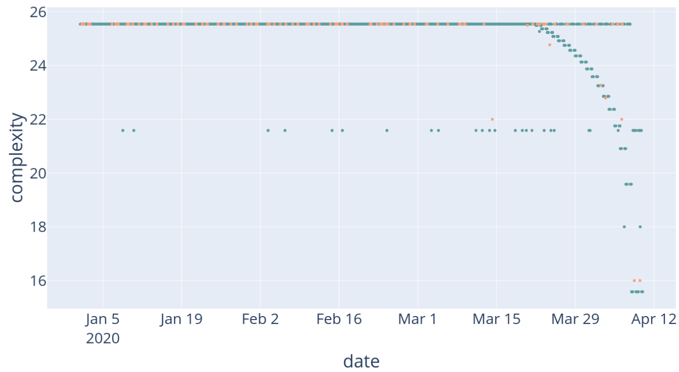
In addition to the number of available predicate concepts, one might also tweak their complexity. In our example scenarios, we used only a handful of predicates, hence we chose to assign all predicate concepts the same bit description length, $\log(|\Pi|)$ (see Equation. 19). When using more predicate concepts, we may instead give different costs to some of them, to take into account the complexity difference between them: for instance, the generic time concept `year` would likely require less bits than a predicate `tun` (an old Mayan time unit, corresponding to roughly 360 days). Similar to the selection of predicates to be used, the complexity assigned to each predicate concept, that is, their encoding, denotes the subjectivity of the user.



(a)



(b)



(c)

Figure 8. The same memory of events, analyzed through three different sets of predicates. Light events are in blue, TV events in salmon. Figure. 8a uses only time-related predicates (days and hours ago), while Figure. 8b adds label predicates alongside with "dark" and day/night predicates. Figure 8c adds the possibility to select directly select any event from the memory of size N, at the cost of $l(N)$ bits.

4. Related Works

Our work is intended to be integrated into larger-scale frameworks to monitor and detect events in complex environment such as smart homes. In these works, the approach to smart homes is often regarded as self-organizing systems [12,13]. As such, they present capacities of adaptation to new goals, new components, new environment. A commonly used approach is the principle of autonomic system, which minimizes user's intervention for management of the system [13,14].

We want to inscribe our work among other classical concurrent approaches to abduction. In situations where more data is available, we could for instance rely on correlation or causal inference from known relations [15,16]. Previous relations between inference and complexity have been studied. In fact, the case of inference was one of the motivations for R. Solomonoff to introduce his universal algorithmic probability [17] as a tool to reach an idealized inference machine, creating the notion of complexity concomitantly to Kolmogorov. Subsequently, notions of complexity re-emerged in causal inference: [18] found, when a causal link exists between two random variables, considering the direct joint probability is simpler, in terms of Kolmogorov complexity, than the inverse direction.

The relation between complexity, compression and causality was used in [19] to devise the PACK algorithm. It models a dataset by using a family of decision trees where each tree describes how one variable can be expressed given the others. By choosing the model minimizing the total description length (i.e. description of the model and description of the errors), PACK compresses the dataset while finding relations between variables which can be further analyzed. More recently, [20] used Minimum Description Length to determine, given a joint probability distribution over (X, Y), whether X causes Y or Y causes X. Their method is based on tree models, and implying that a model respecting the causal relation will be simpler to describe.

Another topic where AIT can provide original approaches is event mining in data streams. [21] provides a good review of modern approaches and techniques in the field. Some previous work can also be noted for having used AIT techniques to qualify and detect events in time series data. For instance, [22,23] propose weighted permutation entropy as a proxy for complexity measures in time series data, and use it to find relations between different time series. [24] proposes a MDL approach to find the intrinsic dimensions of time series. All these approaches are interesting, and can be integrated into our canvas as tools to detect events using only complexity. Using this, one could achieve a purely complexity-driven process for detecting and qualifying events.

The philosophy of our approach can be closely related to the "Isolation forests" method[25,26]. It evaluates the isolation of data points by constructing random binary tree classifiers. On average, outlier points will require less operations to be singled out. Using the average height of leaves in the tree as a metrics, this approach succeeds in identifying outlier points without having to define a "typical" point. This approach can be understood in terms of complexity: each node of binary tree classifier needs a fixed amount of information to be described (which variable and threshold are used). So nodes that place higher need less information to be described. As such, outliers need less information to be singled out. Compared to ours, this method is tailored for data-points living in the same metric space. By using predicates as a proxy for complexity computation, our methods is more general, as it is agnostic regarding the nature of events. However, the introduction of predicates adds a subjectivity in the determination of memorable points, as we will discuss later.

While all these works advocate in favor of a strong link between complexity and the discovery of causes, not other works extends the notion up to the point we propose in this paper, using complexity only as a tool to express the intuitive notion of memorability, and using it for inference.

5. Perspectives

The practical application of the theoretical notions of memorability and relative memorability requires future developments. In this section, we have selected two of them which seem to us, to date, the most challenging.

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 [27, 28]. 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.

6. Conclusion

The ability to identify some events as memorable is useful in the current context of computing, where connected devices record many events of different characteristics, magnitudes, types. This is particularly true in complex cyber-physical systems such as smart homes. We propose an approach to evaluate the memorability as a difference between the expected description complexity of an event and its actual value. With this definition, something is memorable if it requires much less or much more information to be described. To formalize this notion, we used the minimum Minimum Description Length principle exposed in Algorithmic Information Theory. By defining filter operations from predicates and successively applying these filters, we define formal ways of describing events. The length of the shortest of such way can be used as a measure of description complexity.

We provided an implementation algorithm to compute this measure on events and showed some of its results in two smart home examples. These application scenarios qualitatively show how our measure fares in comparision with the human notion of memorability, and how this measure can be used to propose relevant hypotheses in an abductive inference process without having further knowledge of the system.

Finally, we discussed the inherent subjectivity of our approach, embodied in the choice of available predicates and their encoding. We link our approach to existing works which use Algorithmic Information Theory principles to analyze time series data or detect outlier events.

Author Contributions: Conceptualization, É. Houzé and J-L. Dessalles; methodology, software, validation, writing–original draft, É. Houzé; supervision, writing–review and editing J-L. Dessalles, A. Diaconescu and D. Menga. All authors have read and aggreed to the submitted version of the manuscript.

Funding: TODO je ne sais pas quoi mettre ici ? Indiquer le financement de la thèse ?

Data Availability Statement: All code and data used for the experiments can be found at https://github.com/EtienneHouze/memorability_code. The iCasa smart home simulator from the Adele research Group, which was used to generate sensor data, can be found at: http://adeleresearchgroup.github.io/iCasa/snapshot/index.html

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Magnani, L. *Abduction, reason and science: Processes of discovery and explanation*; Springer Science & Business Media, 2011.
2. Strauss, U.; Grzybek, P.; Altmann, G. Word length and word frequency. In *Contributions to the science of text and language*; Springer, 2007; pp. 277–294.
3. Li, M.; Vitányi, P.; others. *An introduction to Kolmogorov complexity and its applications*; Vol. 3, Springer, 2008.
4. Dessalles, J.L. Coincidences and the encounter problem: A formal account. *arXiv preprint arXiv:1106.3932* **2011**.

5. Delahaye, J.P.; Zenil, H. Numerical evaluation of algorithmic complexity for short strings: A glance into the innermost structure of randomness. *Applied Mathematics and Computation* **2012**, *219*, 63–77.
6. Kolmogorov, A.N. Three approaches to the definition of the concept "quantity of information". *Problemy peredachi informatsii* **1965**, *1*, 3–11. Publisher: Russian Academy of Sciences, Branch of Informatics, Computer Equipment and . . . .
7. Murena, P.A.; Al-Ghossein, M.; Dessalles, J.L.; Cornuéjols, A.; others. Solving Analogies on Words based on Minimal Complexity Transformations. International Joint Conference on Artificial Intelligence. IJCAI, 2020.
8. Chater, N.; Vitányi, P. Simplicity: A unifying principle in cognitive science? *Trends in cognitive sciences* **2003**, *7*, 19–22. Publisher: Elsevier.
9. Dessalles, J.L. The Pisa Tower effect.
10. Lalanda, P.; Gerber-Gaillard, E.; Chollet, S. Self-Aware Context in Smart Home Pervasive Platforms. 2017 IEEE International Conference on Autonomic Computing (ICAC), 2017, pp. 119–124. doi:10.1109/ICAC.2017.1.
11. Elias, P. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory* **1975**, *21*, 194–203. doi:10.1109/TIT.1975.1055349.
12. Kramer, J.; Magee, J. A rigorous architectural approach to adaptive software engineering. *Journal of Computer Science and Technology* **2009**, *24*, 183–188.
13. Kounev, S.; Lewis, P.; Bellman, K.L.; Bencomo, N.; Camara, J.; Diaconescu, A.; Esterle, L.; Geihs, K.; Giese, H.; Götz, S.; others. The notion of self-aware computing. In *Self-Aware Computing Systems*; Springer, 2017; pp. 3–16.
14. Kephart, J.O.; Chess, D.M. The vision of autonomic computing. *Computer* **2003**, *36*, 41–50. Publisher: IEEE.
15. Peters, J.; Janzing, D.; Schölkopf, B. *Elements of causal inference: foundations and learning algorithms*; MIT press, 2017.
16. Fadiga, K.; Houzé, E.; Diaconescu, A.; Dessalles, J.L. To do or not to do: finding causal relations in smart homes. 202 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), 2021. _eprint: 2105.10058.
17. Solomonoff, R.J. A formal theory of inductive inference. Part I. *Information and control* **1964**, *7*, 1–22. Publisher: Elsevier.
18. Janzing, D.; Schölkopf, B. Causal inference using the algorithmic Markov condition. *IEEE Transactions on Information Theory* **2010**, *56*, 5168–5194. Publisher: IEEE.
19. Tatti, N.; Vreeken, J. Finding good itemsets by packing data. 2008 Eighth IEEE International Conference on Data Mining. IEEE, 2008, pp. 588–597.
20. Marx, A.; Vreeken, J. Causal inference on multivariate and mixed-type data. Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, 2018, pp. 655–671.
21. Aggarwal, C.C. *Outlier Analysis*; Springer International Publishing, 2017.
22. Batista, G.E.; Wang, X.; Keogh, E.J. A complexity-invariant distance measure for time series. Proceedings of the 2011 SIAM international conference on data mining. SIAM, 2011, pp. 699–710.
23. Fadlallah, B.; Chen, B.; Keil, A.; Príncipe, J. Weighted-permutation entropy: A complexity measure for time series incorporating amplitude information. *Physical Review E* **2013**, *87*, 022911.
24. Hu, B.; Rakthanmanon, T.; Hao, Y.; Evans, S.; Lonardi, S.; Keogh, E. Discovering the intrinsic cardinality and dimensionality of time series using MDL. 2011 IEEE 11th International Conference on Data Mining. IEEE, 2011, pp. 1086–1091.
25. Liu, F.T.; Ting, K.M.; Zhou, Z.H. Isolation Forest. 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 413–422. doi:10.1109/ICDM.2008.17.
26. Hariri, S.; Kind, M.C.; Brunner, R.J. Extended Isolation Forest. *IEEE Transactions on Knowledge and Data Engineering* **2021**, *33*, 1479–1489. doi:10.1109/TKDE.2019.2947676.
27. Dessalles, J.L. From conceptual spaces to predicates. Applications of conceptual spaces: The case for geometric knowledge representation; Zenker, F.; Gärdenfors, P., Eds.; Springer: Dordrecht, 2015; pp. 17–31. doi:10.1007/978-3-319-15021-5_2.
28. Gärdenfors, P. *Conceptual spaces: The geometry of thought*; MIT press, 2004.