Estimations of Integrated Information Based on Algorithmic Complexity and Dynamic Querying

Estimations of Integrated Information Based on Algorithmic Complexity and Dynamic Querying

Alberto Hernández-Espinosa[1], Hector Zenil[*2,3,4,5], Narsis A. Kiani[2,3,4], and Jesper Tegnér[3,6]

[1]Depto. de Matemáticas, Facultad de Ciencias, UNAM, México;
[2]Algorithmic Dynamics Lab, Karolinska Institute (KI);
[3]Unit of Computational Medicine, Center for Molecular Medicine, Department of Medicine Solna, KI, Stockholm, Sweden;
[4]Algorithmic Nature Group, Laboratory of Scientific Research (LABORES) for the Natural and Digital Sciences, Paris, France;
[5]Oxford Immune Algorithmics, Reading, U.K.;
[6]Biological and Environmental Sciences and Engineering Division, Computer, Electrical and Mathematical Sciences and Engineering, Division, King Abdullah University of Science and, Technology (KAUST), Kingdom of Saudi Arabia.

Abstract

Information has emerged as a language in its own right, bridging several disciplines that analyze natural phenomena and man-made systems. Integrated information has been introduced as a metric to quantify the amount of information generated by a system beyond the information generated by its elements. Yet, this intriguing notion comes with the price of being prohibitively expensive to calculate, since the calculations require an exponential number of sub-divisions of a system. Here we introduce a novel framework to connect algorithmic randomness and integrated information, and a numerical method for estimating integrated information using a perturbation test rooted in algorithmic information dynamics. This method quantifies the change in program size of a system, when subjected to a perturbation. The intuition behind is that if an object is random then random perturbations have little to no effect to what happens when a shorter program but when an objects has the ability to move in both directions (towards or away from randomness) it will be shown to be better integrated as a measure of sophistication telling apart randomness and simplicity from structure. We show that an object with a high integrated information value is also more compressible, and is therefore more sensitive to perturbations. We find that such a perturbation test quantifying compression sensitivity, provides a system with a means to extract explanations–causal accounts–of its own behaviour. Our technique can reduce the number of calculations to arrive to some bounds or estimations, as the algorithmic perturbation test guides an efficient search for estimating integrated information. Our work sets the stage for a systematic exploration of connections between algorithmic complexity and integrated information at the level of both theory and practice.

KEYWORDS: integrated information; algorithmic complexity; algorithmic information theory; algorithmic randomness; algorithmic information dynamics; reprogrammability test; causality.

---

[*]Co-first and corresponding author: hector.zenil [at] algorithmicnaturelab [dot] org

1

1 Introduction

The development of techniques to decipher the structure and dynamics of complex systems is a rich inter-disciplinary research area which is not only of fundamental interest but also important in numerous applications. Broadly speaking, dynamical aspects such as stability and state-transitions within such systems have been of major interest in statistical physics, dynamical systems, and computational neuroscience [21, 16, 15]. Here, complex systems are defined by a set of non-linear evolution equations. Cellular automata, spin-glass systems, Hopfield networks, and Boolean networks, have for example been used as numerical experimental model systems to investigate the dynamical aspects of complex systems. Due to the complexity of the analysis, notions such as symmetries in the systems, averaging (e.g. mean-field techniques), and separation of time-scales, have all been instrumental in deciphering the core principles at work in such complex systems. In parallel, network science has emerged as a rich interdisciplinary field, essentially analyzing the structure of real networks in different areas of science and in diverse application domains [12]. Examples include social, biological and electrical networks, the web, business networks and the interconnected internet. By a structural analysis, which has dominated these investigations, we refer to statistical descriptions of network connectivity. Networks can be described globally, in terms ranging from the degree to which they differ from a random Poisson distribution of links, to their modular organization, including their local properties such as local clustering around nodes, special nodes/links with high degrees of betweenness or serving specific roles in the network, and local motif structures. Such network features can be used to classify and describe similarities and differences between what appear to be different classes of networks across and within different application domains. Finally, due to the rich representational capacity of networks and their usefulness across science, technology, and applications, work in machine learning, in particular graph convolutional networks and embedding techniques, is currently making headway in devising ways to map these non-regular network objects onto a format such that machine learning techniques can be used to analyze their properties [13].

Now, we may ask if integrated information theory (IIT) is proposed to be of relevance for the analysis of complex networks, we ask how is IIT related to fundamental questions underpinning research and thinking of complex systems? On the one hand, we find a rich body of work dealing with what could be referred to as technical, computational challenges, and application-driven investigations. For example, which global and local properties should be computed and how to do so in an efficient manner. However, at a more fundamental level we find essentially two challenges, which in our view have a bearing on the core intellectual driving force of complex systems. First: What is the origin of and mechanisms propelling order in complex systems? Secondly, and of major concern for the present paper: Is the whole - in some sense - larger than the sum of its parts? Both questions are vague when formulated in words, as above, but they can readily be technically specified within a model class. The motivation

for the second question is that it appears that there are indeed phenomena in nature which cannot easily be explained only with reference to their parts, but seem to require that we adopt a holistic view. Since Anderson's classic 1972 essay, there has been an animated and at times heated discussion of whether there is anything which could be referred to as emergence.[11]

Tononi and his group have developed a formalism–targeting exactly such a holistic analysis–specifically to quantify the amount of information generated by a system – defined as a set of interconnected elements–beyond the information generated by the parts (subsets) of the system. Their motivation was that in order to develop a theory of consciousness [26]. In that quest, they perceived a necessity to define a measure which could quantify the amount and degree of consciousness, a measure they refer to as ϕ, which in turn constitutes the core of Integrated Information Theory or IIT. Importantly, in the present work we distinguish between the issue of the relevance of ϕ for consciousness versus the technical numerical question of how to calculate ϕ. Here we address the computation of ϕ, as it is potentially a means toward a precise formulation for the possible causal relation between a whole and the parts of a system, regardless of its purported relevance to consciousness. To calculate ϕ, Tononi and collaborators have developed a computational toolbox [25]. Yet, calculating ϕ comes with a severe computational cost, as the calculation scales exponentially with the number of elements in the network. Furthermore, the computation requires knowledge of the transition probabilities of the system, which makes computation of anything larger than small systems of order of one magnitude intractable in practice. The calculation of ϕ requires a division of the system into smaller subsets, ranging from large pieces down to singletons, every division into k pieces can be instantiated in $\binom{N}{k}$ different ways. Using this procedure from Tononi, elements that have small causal influences on the activity of other elements can be identified. A system with low ϕ is therefore characterized by the fact that changes in subsets of the system do not affect the rest of the system. Such a system is therefore considered to be a non-integrated system. This observation entails a key insight, namely, that if a system is highly integrated among its parts, then the different parts can be related to each other, or more precisely, they can be used to describe other parts of the system. Then the parts are in some sense simple and should be compressible.

This is the observation and intuition behind our method, which employs a formalized notion of complexity to exploit this insight and thereby allow a more efficient, guided search in the space of algorithmic distances, in contrast to exhaustive computations of the distance between statistical distributions, as currently implemented in IIT. Technically we are therefore not required to perform a full computation of what is referred to as the input-output repertoire (see Methods for technical details). This, in brief, is our motivation for introducing our method, which is based on algorithmic information dynamics [18, 43, 44]. At its core is a causal perturbation analysis and a measure of sophistication connected to algorithmic complexity. Our approach exploits the idea that causal deterministic systems have a simple algorithmic description and thus a simple

generating mechanism sufficient to simulate and reproduce complex systemic behaviour. Using this technique we can assess the effect of perturbations, and thereby exploit the fact that, depending on the algorithmic complexity of a system, the perturbation will induce different degrees of change in algorithmic space. In short, a system will be highly integrated only if the removal or perturbation of its parts has a non-linear effect on the generative program producing the system in the first place. Interestingly, even Tononi suggested early on that algorithmic complexity could be connected to the computation of integrated information [10]. However, a lossless compression algorithm was used to approximate ϕ. Here we contribute to the formalization of such a suggestion by using stronger tools, which we have recently developed, to approximate complexity. At the core of algorithmic information is the concept of minimal program-size and Kolmogorov-Chaitin complexity [24, 14]. Briefly, the Kolmogorov-Chaitin complexity $K(x)$ of an object x is the length of the shortest computer program that produces x and halts. $K(x)$ is uncomputable but can be approximated from above, meaning one can find upper bounds by using compression algorithms, or rather more powerful techniques such as those based on algorithmic probability [17, 30, 39], given that popular lossless compression algorithms are limited and more closely related to classical Shannon entropy than to K itself [41, 37, 38]. One reason for this state of affairs is that, as demonstrated in [3], there is a fundamental difference between algorithmic and statistical complexity with respect to how randomness is characterised in opposition to causation. Specifically, algorithmic complexity implies a deterministic description of an object (it defines the algorithmic information content of an individual sequence/object), whereas statistical complexity implies a statistical description (it refers to an ensemble of sequences generated by a certain source). Approaches such as transfer entropy [5], Granger causality [19], and Partial Information Decomposition [32, 33] that are based on regression, correlation and/or a combination of regression, correlation and intervention but ultimately relying on probability distributions, fall into this category. Hence for better-founded methods and algorithms for estimating algorithmic complexity, we recommend the use of our tools, which are already being used by independent groups working on, for example, biological modelling [22], cognition [31] and consciousness [29]. These tools are based on the theory of algorithmic probability, and are not free from challenges and limitations, but they are better connected to the algorithmic side of algorithmic complexity, rather than only to the statistical pattern-matching side that current approaches using popular lossless compression algorithms exploit, making these approaches potentially misleading [41]. Our procedure, in brief, is as follows. First, we deduce the rules in systems of interest: we apply the perturbation test introduced in [35, 18, 44] to ascertain the computational capabilities of networks. Next, simple rules are formalized and implemented to simulate the behaviour of these systems. Following this analysis, we perform an automatic procedure, referred to as a meta-perturbation test, which is applied over the behaviour obtained by the aforementioned simple rules, in order to arrive at explanations of such behaviour. We incorporate the ideas of an interventionist calculus (c.f. Judea Pearl [28]) and perturbation analysis within what

4

we call Algorithmic Information Dynamics, and we go beyond pattern identification using probability theory, classical statistics, and correlation analysis by developing a model-driven approach that is fed by data. This contrasts with a purely data-driven approach, and is a consequence of the fact that our analysis considers the algorithmic distance between models.

2 Basic concepts - Integrated Information Theory (IIT)

Integrated information theory (IIT) postulates that consciousness is identical to integrated information and that a system's capacity for consciousness can be expressed by a quantitative measure denoted by ϕ. Tononi defines integrated information as "the amount of information generated by a complex of elements, above and beyond the information generated by its parts" (Consciousness as integrated information: a provisional manifesto. Tononi 2008) and states, *"The integrated information theory* (IIT) of consciousness claims that, at a fundamental level, consciousness is integrated information" (Consciousness as integrated information: a provisional manifesto. Tononi 2008, italics in original). IIT aims to explain "relationships between consciousness and the Physical Substrate of Consciousness (PSC), and starts from essential properties of phenomenal experience, and derives the requirements for the physical substrate of consciousness." [7]

A first formulation consisted in 5 axioms, which are held to be self-evident within IIT:

1. Existence. It is not possible to deny the existence of consciousness, and that consciousness exists is self-evident from within its own perspective.

2. Composition. Each experience has components.

3. Information: An experience is specific to certain things, distinct from others.

4. Integration. Consciousness is irreducible to separate elements.

5. Exclusion. Because consciousness specifies certain things, it excludes others; at the same time, it flows with a specific velocity.

From these axioms IIT formulates five postulates, in essence making causal claims:

1. The existence of consciousness implies a system of mechanisms with a particular cause-effect power.

2. The composite nature of consciousness implies that its system's mechanistic elements must have the capacity to combine, and that those combinations have cause-effect power.

3. Because consciousness is informative, it must specify, or distinguish one experience from another.

4. Because consciousness is integrated, every one of its element must have the capacity to act as a cause on the rest of the system and to be affected by the rest of the system. If a system can be divided into two parts without affecting its cause-effect structure, it fails to satisfy the requirement of this postulate.

5. Because of its exclusivity, the various simultaneous subsets of mechanisms in a system have varying cause-effect structures.

Since the postulates make causal claims, it is paramount to establish a quantitative procedure to assess whether or not such a cause-effect structure is present. Thus, is the system integrated with respect to its internal cause-effect organization? To this end, IIT introduces the parameter ϕ as a measure of the amount of integrated information and hence the level of consciousness of the system. The measure ϕ is calculated by dividing the set of elements of a model of the system progressively into subsets in order to identify elements whose (de)activation has few causal consequences upon the (de)activation of other elements. Then, if no logically possible partition of the system results in a loss of connection, the conclusion is that the system has no positive ϕ value. In other words, changes in subparts of a system that do not affect the rest of the system identify a non-integrated system.

A ϕ estimation calculus

The integrated information theory defines integrated information (ϕ) as the effective information of the minimum information partition (MIP) in a system (Tononi, 2004, 2012; Oizumi et al., 2014, 2016a; Tononi et al., 2016). The MIP is also defined as the partition having minimum effective information among all possible partitions.

$$\phi[X; x] =: \varphi[X; x, MIP(x)]$$

$$MIP(x) =: argmin\varphi(X; x, P)$$

Where X is the system, x is a state, and P is a partition $P = M_1, \ldots, M_r$.

Importantly, identifying the MIP requires searching all possible partitions and comparing their effective information ϕ. This effective information is specified in terms of effect and causal information, that is, the distance between two probability distributions: one for the unpartitioned (unconstrained) partition (this can be the full set of nodes of the whole system or one of its possible partitions) and a partition of this latter. Such probability distributions determine probabilities of all possible future (effect) or past (causal) states of an arbitrary partition being in a current state. This means that comparing one set of nodes that can be the full set of nodes of the system or a subset (partition) of itself with all possible partitions of this set of nodes, MIP represents the partitions

with the minimal value of the distance between probability distributions of the set of nodes and one of all its possible partitions.

When a set of nodes is chosen to compute effective information, this is referred to as a 'mechanism', and the partition to which it is compared is referred to as the 'purview'. The distance between probability distributions is computed by means of an adaptation of the Earth Mover's Distance (EMD) algorithm, which is a method to evaluate dissimilarity between two multi-dimensional distributions in a given feature space where a distance measure between single features, which we call the ground distance, is given. The EMD "lifts" this distance from individual features to full distributions. Note that EMD is referred to as a Wasserstein metric in mathematics, and is commonly used in machine learning as a natural metric between two distributions [8].

Intuitively, given two distributions, one can be seen as a mass of earth properly spread in space, the other as a collection of holes in that same space. Then, the EMD measures the least amount of work needed to fill the holes with earth. Here, a unit of work corresponds to transporting (by an optimal transport method) a unit of earth a unit of ground distance.

3   Methods

In this section we introduction of the meta-perturbation analysis, additional technical details of which are presented in the Appendix. Next, we recap the causal perturbation and causal analysis leading up to the notion of program-size divergence, which is our core metric for how different programs, i.e. systems–more or less integrated–respond to perturbations

3.1   Programmability test and meta-test

In [35]a programmability test is introduced which was inspired by the Turing test, while being based on the view that the universe and all physical systems living in it and able to process information can be considered (natural) computers [35] equipped with particular computational capabilities [34].

The programmability test is explained as: "...replacing the question of whether a system is capable of digital computation with the question of whether a system can behave like a digital computer and whether a digital computer can exhibit the behaviour of a natural system."

Then, in the same way that the Turing test proceeds to ask questions of a computer in order to determine whether it is capable of computing an intelligent behaviour, the programmability test aims to know what a specific system is capable of computing by means of algorithmic querying [45].

In practice, the programmability test is a system perturbation test [35, 36] that "asks" questions of a computational system in the form: *what is your output (answer) given this question (input)?*. This idea is applied to ϕ_K's implementation so that once the set of all possible answers of a system is obtained, this set is analyzed and generalized to deduce the rules that should not just offer a

picture of its computability capabilities, but also simulate and give an account of the behaviour of the system itself.

A second step after this perturbation test is to analyze its results in order to construct a computer program–as simple as possible–capable not only of reproducing the output repertoire but also of giving an account of the programmability capabilities of the system itself, that is, rules capable of producing a certain output given an input, and at the same time explaining where, in ordinal terms, such an output could be placed relative to the order of the full output repertoire. This latter aspect we refer to as the meta-perturbation test.

Then, ϕ_K not only applies a perturbation test over a system, but also a meta-perturbation test over results obtained on the first test. The rules found in this meta-test are used not only as compressed specifications or representations of the behaviour itself, but also as rules that give a sort of account of the behaviour of the system.

This can be done because the systems analyzed in IIT are well known, or in other words, since all node-by-node operations are well defined, it is easy to compute all possible outputs (answers) for all possible inputs (questions or queries), corresponding to what in IIT are referred to as repertoires. In the context of ϕ_K, a meta-test is applied in order to find the rules that describe the behaviour embedded in repertoires of a system, instead of trying to ascertain the rules that define how the system works.

A system specified in this manner turns on a "computer", recording it's own behaviour (e.g the repertoires) as well as probing itself, e.g. the action of ϕ_K, in such a manner as to potentially give an account of its own behaviour. To make this possible a system specification must be enabled with an explanatory interface based on these simple embedding behaviour rules. ϕ_K goes beyond the original ϕ in that the programmability test searches for the rules underlying the behaviour of a system rather than generating a description of its possible causal connections. While in IIT these rules are defined a priori and induced by perturbation, ϕ_K's objective is not only to find rules that simulate, but also describe such behaviour in a brief manner (thus simple rules) and make predictions about the behaviour of the system. The field of Algorithmic Information Dynamics [18, 44] implements this approach by asking what changes to hypothesized outputs mean for the hypothesized underlying program generating an observation, after an induced or natural perturbation.

The simple rules discovered and used for the calculation of ϕ_K are used here exclusively to compose *constrained/unconstrained distributions* used in IIT for obtaining *cause-and-effect information*, a key concept from which the integration of information derives. The rest of the calculus–earth mover's distance measurements, the calculus of conceptual spaces, major complex and finding the MIP– remains as specified in IIT 3.0.

3.2   Causal perturbation analysis

From a statistical standpoint, it would be typical to suggest that the behaviour of two time series, let's call them X and Z, would potentially be causally con-

nected if they were statistically correlated. Yet, there are several other cases that would not be distinguishable after a correlation test. A first possibility is that the time series simply shows similar behaviour without being causally connected, i.e. there is a shared upstream causal driver Y, concealed from the observer. Another possibility is that they are causally connected, but that correlation does not tell us whether it is a case of X affecting Z, or vice versa.

Perturbation analysis allows some disambiguation. The idea is to apply a perturbation on one time series and see how the perturbation spreads to the other time series. Perturbing the data-point in position 5 the time series Z as shown in Figure 1 multiplying it by -2, X does not respond to the perturbation. This means that for this data point, X remains the same. This suggests that there is no causal influence of Z on X.
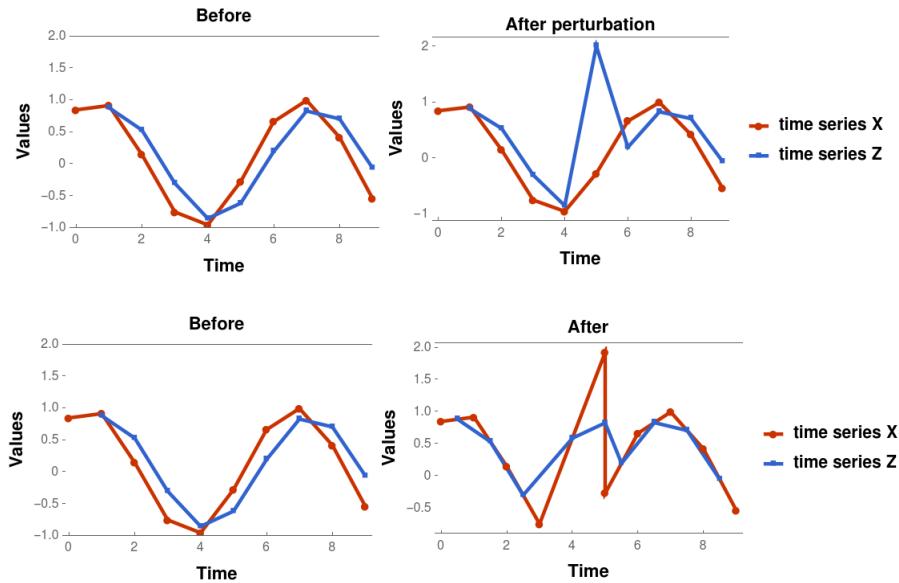
Figure 1: Causal intervention analysis on time series X and Z before and after perturbation in Z (top) and X (bottom). The values of Z come from the moving average of X, so there is a one-way causal relationship: perturbing X has an effect on Z but perturbing Z has no effect on X thereby suggesting the causal relationship.

In contrast, if the perturbation is applied to a value of X, Z changes and follows the direction of the new value, suggesting that the perturbation of X has a causal influence on Z. From behind the scenes, we can reveal that Z is the moving average of X, which means that each value of Z takes two values of X to calculate, and so is a function of X. The results of these perturbations produce evidence in favour of a causal relationship between these processes, if we did not know that they were related by the function we just described.

9

This suggests that it is X which causally precedes Z. So we can say that this single perturbation suggests a causal relationship illustrated in Figure 2.

Figure 2: Possible self-loopless causal relationship between two unlabelled variables X and Z.
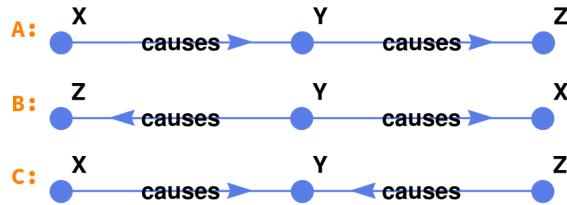
Figure 3: Acyclic path graphs representing all possible self-loopless connected causal relationships among 3 unlabelled variables.

There are a number of possible types of causal relationship between three events (see Figure 3) that can be represented in what is known as a directed acyclic graph (DAG), that is, a graph that has arrows implying a cause and effect relationship but has no loops, because a loop would make a cause into the cause of itself, or an effect that is also its own cause, something that would be incommensurate with causality. In these graphs, nodes are events and events are linked to each other if there is a direct cause-and-effect relation. In the first case, labelled A in orange, the event X is the cause of event Y, and Y is the cause of event Z, but X is said to be an indirect cause of Z. In general, we are, of course, always more interested in direct causes, because almost anything can be an indirect cause of anything else. In the second case B, an event Y is a direct cause of both Z and X. Finally, in case C, the event Y has 2 causes, X and Z. With an interventionist calculus such as the one performed on the time series above, one may rule out some but not all cases, but more importantly, the perturbation analysis offers the means to start constructing a model explaining the system and data rather than merely describing it in terms of simpler correlations. In our approach to integrated information, the idea is to identify the set of most likely generating candidates able to produce certain observed behaviour even if such behaviour may not carry any statistical regularity and for all purposes appear statistically random [9]. Strictly speaking, computational mechanics [4, 3], is a framework that bridges statistical inference and stochastic modelling that suggests a model based on an automaton called an ϵ-machine. However, such machines are stochastic in nature and, if the methods used to reconstruct such machines rely on statistical methods, the result is only an apparent causal

representation with no correspondence between internal states and alleged states of the phenomenon observed. In contrast, approaches based on algorithmic probability as approached by algorithmic information dynamics can complement computational mechanics as they provide means to construct non-stochastic automata models that are independent of probability distributions and are in a strict sense optimal and universal [6].

In the case of our two time series experiments, the time series X is produced by the mathematical function $f(x) = Sin(x)$, and thus $Sin(x)$ is the generating mechanism of time series X. On the other hand, the generating mechanism of Z is $MovAvg(f(x))$, and clearly $MovAvg(f(x))$ depends on $f(x)$, which is $Sin(x)$, but $Sin(X)$ does not depend on $MovAvg(f(x))$. In the context of networks, the algorithmic-information dynamics of a network is the trajectory of a network moving in algorithmic-information space together with the identification of those elements that shoot the network towards or away from randomness.

3.3   Causal influence and sublog program-size divergence

According to Algorithmic Information Dynamics [18, 44] there is an algorithmic causal relationship between two states s_t and $s_{t'}$ of a system M and M' if

$$|K(M_{s_t}) - K(M'_{s_{t'}})| \leq \log_2(t) + c$$

That is, if the descriptions of such systems can be bounded by $\log_2$ and a small constant c, then M is most likely equal to M' but in some other time state.

In other words, if there is a causal influence of s_t on s_{t+1} or s_{t+1} on s_t as a system in isolation, their M and M' short descriptions should not differ by more than the description of the difference.

However, if the descriptions of the states of a system (which may be two systems) in different alleged state times are not causally connected, their difference will diverge beyond above bound. In integrated information, causal influence among its parts is what is claimed to be measured and how different elements of a system can be explained by a single model or the other parts of the system informs us as to how integrated a system may be. A system characterized by large divergence is less integrated compared to a system which evolves with small differences in its respective subpart descriptions.

We will suggest that perturbations have to be algorithmic in nature because they need to be made or quantified at the level of the generating mechanisms from the whole or different parts of the integrated system and not at the level of the observations. For example, some $n-$ary expansions of the mathematical constant π according to BPP (named after Bailey-Borwein-Plouffe) formulas [1] allow perturbations to the digits that do not have any further effect because no previous digits are needed to calculate any other segment of π in the same base. The constant π then can be said to be information disintegrated to the extent of the BPP representations. Algorithmically low complexity objects have low integrated information. Similarly, highly random systems have low integrated

information, because perturbations have little to no impact. Integrated information is, therefore, a measure of sophistication that separates high integration from both random and trivially non-random states.

3.4   A simplicity versus complexity test

With the previous section in mind we can proceed to introduce the idea of ϕ_K, where K stands for the letter often used for algorithmic (from Kolmogorov or Kolmogorov-Chaitin) complexity, and ϕ for the traditional of integrated information theory [26]. The measure ϕ_K mostly follows methods that Oizumi and Tononi set forth in [26], where integrated information is measured, roughly speaking, as distances between probability distributions that characterize a MIP (Minimum Information Partition), that is, "the partition of [a system] that makes the least difference"[26].

However, the difference between IIT's ϕ and ϕ_K lies in how ϕ_K circumvents what is called the "intrinsic information bottleneck principle" [27] that traditionally requires an exhaustive search for the MIP among all possible partitions of a system, a procedure responsible for the fact that integrated information computation requires super-exponential computational resources. In contrast to ϕ, which follows a statistical approach to estimating and exhaustively reviewing repertoires, the approach to ϕ_K is based on principles of algorithmic information.

Discovering the simple rules that govern a "discrete dynamical system" [25] like those studied in IIT presupposes the analysis of its general behaviour in pursuit of a dual agenda: first, to determine its computational capabilities, and secondly to obtain explanations and descriptions of the behaviour of the system.

As a consequence, one of the major adaptations of IIT is that ϕ_K uses the concept of Unconstrained Bit Probability Distribution (UBPD), that is, the individual probabilities associated with a node of a system taking values of 1 (ON) or 0 (OFF) after it has been "fed" all its possible inputs or after a perturbation.

In the context of ϕ_K, UBDP is computed using simulation and definition systems governed by simple rules, unlike ϕ, which uses the TPM (Transition Probability Matrix) to compute IIT's unconstrained/constrained probability distributions.

In Figure 4 and Table 1 the concept UBPD and its calculus is explained, using the example used by Oizumi et. al. in [26].

In order to explain the notion of UBPD, in Figure 4 we use Oizumi's example used in [26] to calculate information integration. Figure 4-A shows the network representation: three nodes fully connected with different types of operation executed on its inputs, that is, for example, inputs to node A (coming from B and C nodes) will be processed in a logical OR operation. In Figure 4-B the adjacency matrix that represents the same same network is shown. This adjacency matrix uses the number 1 to indicate if a node receives signals (inputs) for another node. For example, the first row in the adjacency matrix indicates that node 1 or A receives inputs from nodes B and C, denoted as nodes 2 and

```
1  am = {{0, 1, 1}, {1, 0, 1}, {1, 1, 0}};
2  dyn = {"OR", "AND", "XOR"};
3  calcUBPOutputs[1,am,dyn]//AbsoluteTiming
4  calcUBPOutputs[2,am,dyn]//AbsoluteTiming
5  calcUBPOutputs[3,am,dyn]//AbsoluteTiming
```

```
1  {0.000575,<|"ZeroProb"−>0.25,"OneProb"−>0.75|>}
2  {0.000287,<|"ZeroProb"−>0.75,"OneProb"−>0.25|>}
3  {0.000341,<|"ZeroProb"−>0.5,"OneProb"−>0.5|>}
```

Table 1: Computing UBPD for system shown in Figure 1. Lines 1, 2: Definition of the system in Figure 1 by adjacency matrix (line 1) and dynamics (line 2). Line 3-5: Calculation of individual probabilities that each node of the system will take values 0/1 across the whole output repertoire. Results square: time of computation in seconds and UBPD distribution.

3. Finally, Figure 4-C shows the full input and output repertoires, that is, for the full set of all possible inputs to this system, all corresponding outputs are calculated according to the logical operations defined.

Table 1 shows code for computing UBPD for the system in Figure 4. This computation starts with the specification of the adjacency matrix (line 1) and internal dynamic (line 2) of the target system. Then, lines 1 and 2 in Table 1 represent code to network specified in Figures 4-A and 4-B.

In the IIT approach, the system is perturbed with all possible inputs to obtain the full output repertoire (Figure 4-C). Then, in the context of ϕ_K, UBPD corresponds to the distribution of probabilities that each node will take values 0/1 in the output/input repertoires after the perturbation. For instance, in Figure 5-A, full input and output repertoires are shown for network in Figure 4-A. Now, let's say we want to compute the future probability distribution, that is, the probability necessary to compute effect information according to [26]. In this case we take output repertoire as a reference and we compute the probability of nodes in the future (outputs) taking the values 0 or 1. For node A, for example, the probability that node A takes the value of 1 is 0.25, that is 2/8, and that it takes the value of 0 is 0.75 or 6/8. These values are called the UBPD for node A. A resume of UBPD for all nodes is given in Figure 5-B.

Once UBPD is computed for a subject partition, in this case the full system's probability distribution is computed by multiplying UBDPs. Let's take as an example the future probability of input {0, 0, 0}, computed as $P(A) = 0$ * $P(B) = 0$ * $P(C) = 0$, that is, 0.25 * 0.75 * 0.5 (see first row in Figure 5-C). When all future probabilities are computed in this manner, the result is the distribution shown in Figure 5-D, which is exactly the same one computed in [26], as shown in Figure 5-E.

In general, UBDP is used to compute probability distributions of a system in the context of ϕ_K, which mirrors the "constrained/unconstrained probability
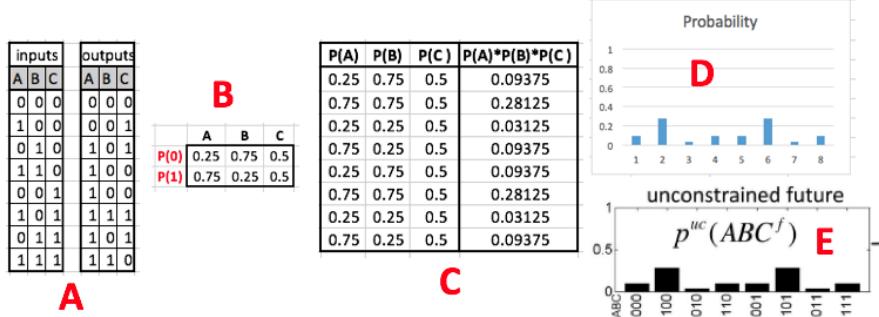
Figure 4: An example of using UBPC to calculate unconstrained output distribution.

distributions" in [26], that is, probability distributions of input/output patterns for specific configurations (partitions) of the system, in contrast to what IIT 3.0 does. In this last case, Mayner shows how probability distributions are computed in the context of IIT in his S1 text mentioned in [25], using terms such as "marginalization" and "virtual elements" that seem to be highly complex methods.

Then, in the context of ϕ_K, UBDP aims to obtain the same results in terms of probability distributions, in a manner equivalent to IIT but by following a different conceptual approach. Our measure ϕ_K uses adapted methods, having algorithmic complexity as a background, to compute information integration.

In Table 1, lines 3-5 shows Mathematica code that computes UBDP for the system specified in lines 1 and 2, that is, by means of an adjacency matrix and an array of computations that nodes perform, or the system dynamics. Table 1 also shows results of this computation in this order: 1) time needed to compute, followed by probability that a node take the value zero (zeroProb) or the value one(oneProb). One can see how the results in Table 1 correspond to UBDP values shown in Figure 5-B.

We should note that for ϕ_K, computation of probability distributions seems to be a task of counting, which for huge systems would be extremely difficult or even impossible, if attempted in a classical/brute force way. But, two important facts should be pointed out here: 1) In the context of ϕ_K, UBPD is not calculated in this traditional way, but is calculated using the simulation of the behaviour of a system represented by a set of simple rules. Then for ϕ_K, an exhaustive review of repertoires is not needed to compute the individual probabilities shown in Table 1, and 2) despite strong theoretical and methodological differences between them, ϕ_K and ϕ lead to the same results.

In the next sections we derive simple rules of a system, using the perturbation test and its application to implement ϕ_K.

14

4 Numerical Results

4.1 Compression sensitivity as informative of integration

To understand the relationship between IIT and algorithmic complexity, we shall briefly move away from the case of networks and focus on binary files and the binary programs that may generate them, the programs that are natural computable candidate models explaining the data. To illustrate the connection, let us take some extreme cases. Let's say we have a random file:

```
In:= randomfile=RandomChoice[CharacterRange["a", "z"],100]
Out= {l,q,l,d,d,x,f,e,u,l,u,b,m,y,a,l,b,l,v,a,v,v,t,l,h,o,x,d
     ,i,a,z,t,z,b,r,y,v,h,c,n,s,u,l,u,w,w,e,a,i,h,h,w,f,d,d,k,o
     ,c,a,k,u,x,v,n,v,e,c,r,t,c,g,s,g,x,y,t,c,h,k,w,c,t,y,u,e,k
     ,y,v,a,h,t,k,f,y,c,r,b,y,y,x}

In:= Compress[randomfile]
Out= 1:eJx1kEEOgyAQRWnSi/Q+XbUnEAUhtJoKInp6zbB5CenmZTL8mf+
Hh55fdlBKxfuFp4/pfbuKj/CHegCL0AqNcIWy1lr4Fe7CDhqNOuM1g0zihDP
caxKP2QNTB1wWZMjY1gsnYfxzxQYaeHnscdDY5q8CkvfYEOBSkG1CbTC14L
raGZF8xJ69UTo4bugn6Fc4hubHOuxJ0FgomVOjX1lOUdJPvg==

In:= Length[Characters@Compress[randomfile]]
Out= 222
```

So, using the Compress algorithm, the resulting compressed object is even longer, this is because the compression algorithms inserts the decompression instructions together with the checksum which ends up increasing the size of the resulting object if the object was not long and compressible enough to begin with.

This is what happens if we perform a couple of random perturbations to the uncompressed file:

```
In:= mutatedfile=ReplacePart[randomfile,{5->"k", 12->"x"}]
Out= {l,q,l,d,k,x,f,e,u,l,u,x,m,y,a,l,b,l,v,a,v,v,t,l,h,o,x,d
     ,i,a,z,t,z,b,r,y,v,h,c,n,s,u,l,u,w,w,e,a,i,h,h,w,f,d,d,k,o
     ,c,a,k,u,x,v,n,v,e,c,r,t,c,g,s,g,x,y,t,c,h,k,w,c,t,y,u,e,k
     ,y,v,a,h,t,k,f,y,c,r,b,y,y,x}
```

The difference between the original and perturbed files is:

```
In := SequenceAlignment[randomfile, mutatedfile]//Column
Out= {l,q,l,d} {{d},{k}} {x,f,e,u,l,u} {{b},{x}} {m,y,a,l,b,l,
     v,a,v,v,t,l,h,o,x,d,i,a,z,t,z,b,r,y,v,h,c,n,s,u,l,u,w,w,e,
     a,i,h,h,w,f,d,d,k,o,c,a,k,u,x,v,n,v,e,c,r,t,c,g,s,g,x,y,t,
     c,h,k,w,c,t,y,u,e,k,y,v,a,h,t,k,f,y,c,r,b,y,y,x}
```

The files only differ by 2 characters, which can be counted using the following code:

```
In := Total[Length/@First/@Select[SequenceAlignment[randomfile
    ,mutatedfile],Head[\#[[1]]]==List\&]]
```

That is, 2/100 or 0.02 percent.

On the other hand, let's take a simple object consisting of the repetition of a single object, say the letter e:

```
In := simplefile=Table["e",100]
Out = {e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e
    ,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e
    ,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e
    ,e,e,e,e,e,e,e,e,e,e,e,e,e,e}
```

A shortest program to generate such a file is just:

```
Table["e",  100]
```

In other languages this could be produced by an equivalent 'For' or 'Do-While' program. We can now perturb the program again, without loss of generality. Let's allow the same 2 perturbations to the data only, and not to the program instructions (we will cover this case later). The only places that can be modified are thus 'e' or 1 instead of 5, say: Table["a",500]

```
In := Length/@First/@Select[SequenceAlignment[Table["a",500],
    Table["e",100]],Head[#[[1]]]==List&]
Out = {500}
```

Now, the original and decompressed versions differ by 500 elements, and not just a small fraction (compared to the total program length) as in the random case. This will happen in the general case with random and simple files; random perturbations will have a very different effect on each case.

An object that is highly integrated among its parts means that one can explain or describe part of each part with some other part when the object is algorithmically simple; then these parts can be compressed by exploiting the information that the said other parts carry over from yet others, and the resulting program will be highly integrated only if the removal of any of these parts has a non-linear effect on its generating program. In a random system, no part contains any information about any other, and the distribution of the individual algorithmic-content contribution of each element is a normal distribution around the mean of the algorithmic-content contributions, hence poorly integrated and trivial. So integrated information is a measure of sophistication, filtering out simple and random systems, and only ascribing high algorithmic information content to highly integrated information systems.

The algorithmic information calculus thus consists of a 2-step procedure to determine:

1. The complexity of the object (e.g. string, file, image)

2. The elements in that object that are less, more, or not sensitive to perturbations that can 'causally steer the system,' i.e. causally modify an object in a surgical algorithmic fashion rather than on the basis of guesswork based on statistics.

Note that this causal calculus is semi-computable, and one can perform guiding perturbations based upon approximations [18, 44]. Also note that we did not cover the case in which the actual instructions of the program were perturbed. This is actually just a subcase of the previous case, that separates data from program. For any program and data, however, we conceive an equivalent Turing machine with empty input, thus effectively embedding the data as part of its instructions. Nevertheless, the chances of modifying the instruction Print[] in the random file case are constant, and for the specific example are: $7/107 = 0.0654$. While for the non-random case, the probability of modifying any piece of the Table[] function is: $8/12 = 0.666667$. Thus, the break-up of a program of a highly causally generated system is more likely under random perturbations.

Notice similarities to a checksum for, e.g., file exchange verification (e.g. from corruption or virus infection for downloading from the Internet), where the data to be transmitted is a program and the data block to check is the program's output file (which acts as a hash function).

Unlike regular checksums, the data block to check is longer than the program, and the checking is not for cryptographic purposes. Moreover, the dissimilarity distance between the original block (shared information) and the output of the actual shared program provides a measure of both how much the program was perturbed and the random or nonrandom nature of the data compressed by the program. And just like checksums, one cannot tamper with the program without perturbing the block to be verified (its output), without significantly changing the output (block) if what the program has encoded is nonrandom and therefore causally/recursively/algorithmically generated. Of course all the theory is defined in terms of binary objects, but for purposes of illustration and with no loss of generality we have shown actual programs operating on larger alphabets (ASCII). And we also decided to perform perturbations on what seems to be the program data and not the program itself (though we have seen that this distinction is not essential) for illustration purposes, to avoid the worst case in which the actual computer program becomes non-functional.

Yet, this means that the algorithmic calculus is actually more relevant, because it can tell us which elements in the program break it completely and which ones do not. But what happens when changes are made to the program output and not the program instructions? Say we exchange an arbitrary e for an a in our simple sequence consisting of a single letter, e.g. the third entry ('a' for 'e'):

If we were to look to the generating program of the perturbed sequence, this would need to account for the a, e.g.

```
In := ReplacePart[Table["e",100],3->"a"]
Out = {e,e,a,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e
      ,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e
```

```
        ,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e
        ,e,e,e,e,e,e,e,e,e,e,e,e,e,e}
```

where the second program is longer than the original one, and has to be, if the sequence is simple, but the program remains unchanged if the file is random because the shortest program of a random sequence is the random sequence itself, and random perturbations keep the sequence random. Furthermore, every element in the simple example consisting of repetitions of e has exactly the same algorithmic content contribution when changed or removed, as all programs after perturbation are of the form:

```
1  ReplacePart[Table["e",100],n−>x]
```

Notice also how this is related to ϕ and possibly any measure of integrated information based on the same principles.

We can now apply all these ideas to the language of networks, with respect to which IIT has, for the most part, been defined. We have shown before that networks with different topologies have different algorithmic complexity values [46], in accordance with the theoretical expectation. In this way, random ER graphs, for example, display the highest values, while highly regular and recursive graphs display the lowest [42]. Some more probabilistic, but yet recursively generated graphs are located between these 2 extremes [40]. Indeed, the algorithmic complexity K of a regular graph grows by $O(logN)$, where N is the number of nodes of the graph, as in a highly compressible complete graph. Conversely, in a truly random ER graph, however, K will grow by $O(\log E)$, where E is the number of edges, because the information about the location of every edge has to be specified.

In what follows we will perform some numerical tests strengthening our analytic derivations.

4.2   Finding simple rules in complex behaviour

A perturbation test is applied to systems which IIT is interested in. The set of answers is analyzed in order to find the rules that 1) make it possible to simulate the behaviour of the system, 2) define their computability power, that is, rules that give an account of what the system can and cannot compute, and 3) rules able to describe and predict behaviour of the same system. The following procedure was applied to estimate ϕ_K.

1. The perturbation test was applied to systems used in IIT to obtain detailed behaviour of the systems.

2. Results in step one were analyzed in order to reduce the dynamics of a system to a set of simple rules. That is, in keeping with the claims of natural computation, we found simple rules to describe a system's behaviour.

3. Rules found in step 2 were used to generate descriptions of what a system is or is not capable of computing and under what initial conditions, without having to calculate the whole output repertoire.

4. A combination of rules found in steps 2 and 3 was used to develop procedures for predicting the behaviour of a system, that is, whether it is possible to have reduced forms that express complex behaviour. Knowing what conditions are necessary for the system to compute something, it is possible to pinpoint where in the whole map of all possible inputs (questions) of a system such conditions may be found.

5. Once rules in steps 2 and 3 are formalized, ϕ_K was turned into a kind of interrogator whose purpose was to ask questions of a system about its own computational capabilities and behaviour.

This kind of analysis allowed us to find that the information distribution in the complex behaviour of systems analyzed in IIT followed a distribution replicated at several scales that is usually and informally identified as 'nested' or 'fractal', and means that it is susceptible of being summarized in simple rules by iteration or recursion, just as is the case with fractals proper. These properties are used to find compressed forms to express answers given by a system when asked for explanations of its own behaviour.

This means that, as noted before, ϕ_K does not compute the whole output repertoire for a system but uses simple rules to express the whole behaviour of the system. Interestingly, the way in which we proceed appears to be connected to whether or not the system itself can explain its behaviour, or rather whether it can see itself to be capable of producing its behaviour from an internal experience (configuration) which is then evaluated by an observer. So ϕ_K takes the form of an automatic interrogator that, in imitation of the perturbation test, asks questions of the form *are you capable of this specific configuration? (pattern), and if so, say where, in the map of the behavioural repertoire, I can find it.*

The benefit of representing systems using simple rules is that it allows an alternative calculation closer to algorithmic complexity and the potential to reduce the number of calculations to derive an educated estimation as compared to the original version of IIT 3.0.

At this point, it is not possible to explain how simple rules define a system in the context of ϕ_K without talking about the pattern of distribution of information in the behaviour of systems like those studied in IIT.

4.3   Simple rules and the pattern of distribution of information

As shown in [45], despite deriving from a very simple program, without knowing the source code of the program, a partial view and a limited number of observations can be misleading or uninformative, illustrating how difficult it can be to reverse-engineer a system from its evolution in order to discover its generating code [45].

In the context of IIT, when we talk about a complex network we find that there are different levels of understanding complex phenomena, such as knowing the rules implemented by each node in a system and finding the rules that

describe its behaviour over time. To achieve the second, as perhaps could be done for the "whole [of] scientific practice" [45], we found it useful to perform perturbation tests in order to deduce the behaviour of the subject systems. Results were analyzed and a pattern in the distribution of information was found to characterize the behaviour of these kinds of systems. Then, as was to be expected, replicating behaviours were amenable to being expressed with simple formulae.

In order to explain how simple rules were found and implemented in ϕ_K, consider as an example the 7-node system shown in Figure 6 whose behaviour is computed by perturbing the system on all possible inputs. The results, or the whole output repertoire, is shown in Table 6 in Appendix A.

Dynamic	Nodes	1	2	3	4	5	6	7
AND	1	0	0	1	0	0	0	1
OR	2	0	0	1	0	0	1	0
OR	3	1	0	0	0	1	0	1
AND	4	1	0	1	0	1	0	1
OR	5	0	0	1	1	0	1	1
OR	6	1	1	1	0	0	0	0
AND	7	0	1	0	1	1	1	0
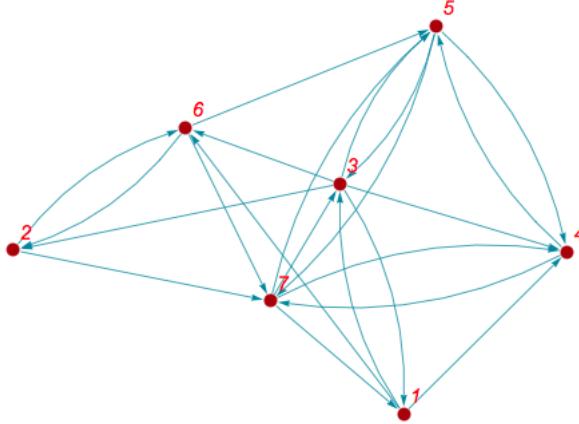
Figure 5: Seven-node system. Adjacency matrix and network representation.

The strategy adopted to find rules that govern a system's behaviour is the same used in almost any branch of science, which is to say we separately observe the behaviour of some of the components of a phenomenon, in this case nodes, while bearing in mind that this behaviour is not isolated but rather the by-product of interacting elements, or in other words, we observe individual behaviours without losing sight of the whole.

When we observe the whole behaviour of the system shown in Figure 6 (see Appendix A, Table 6), we notice mostly chaotic behaviour but with subtle

```
1  analysis07 [[ All , 4]]
```

```
1  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1}
```

```
1  analysis07 [[ All , 5]]
```

```
1  {0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

Table 2: Isolated outputs for nodes {4} and {5} in system introduced in Figure 6 after perturbation.

repetitions of certain patterns.

When the behaviour of elements is isolated, the picture appears clearer. For example, Table 2 shows the isolated behaviour of nodes {4} and {5} of the same subject system.

In Table 2, the isolated behaviour of two nodes of the system in Figure 6 is shown, where it is possible to observe that isolated behaviours for {4} and {5} follow a sort of order. Such patterns are summarized in what we call behaviour tables, shown in Figure 7.

Lowest rows in behaviour tables shown in Figure 7 (within braces) correspond to a compressed representation of behaviours shown in Table 2.

Compressed expressions of behaviour for node {4}, for instance, means: 85 repetitions of digit 0, followed by a pattern repeated 2 times, this pattern being: 1 once, followed by one 0 (that is {1->1, 1->0}). This last pattern is followed for four digits 0, and so on. Here notice that the first number 85 is the sum of the numbers shown in the 3rd column of its behaviour table.

For node 5, a compressed representation of the behaviour means: four times digit 0, followed by four digits 1. This pattern is followed by eight repetitions of digit 1. The last pattern is followed for 94 ($32 + 64$) repetitions of digit 1.

The representation used in this isolation of behaviours is expressed in terms of the nodes that "feed" into target nodes of this example (Node column), namely nodes {4, 5} whose inputs, according to Figures 6 and 7 are: for node {4}:

OVER CURRENT STATE (NODE 4. AND)			
Node	**node-1=pow**	**2^(pow-1)**	**(n+1)/n**
1	0	1	
3	2	4	4
5	4	16	4
7	6	64	4
85->0,{{{1->1,1->0}->2,4->0}->2},16->0			
{{{1->1,1->0}->2,4->0}->2			

OVER CURRENT STATE (NODE 5. OR)			
Node	**node-1=pow**	**2^(pow-1)**	**(n+1)/n**
3	2	4	
4	3	8	2
6	5	32	4
7	6	64	2
{{{4->0,4->1},8->1}->2},32->1,64->1}			

Figure 6: Behaviour Tables for seven-node system in Figure 6. (A) node 4, (B) node 5. From left to right, **Node** column lists input-nodes that feed the target node. $node - 1 = power$ column computes the power used to transform a pattern in the world of the 7-node systems from binary to decimal. **2^pow** column is the result of the binary to decimal transformation operation. The fourth column contains divisions between elements indexed by $n + 1$ in column 3 divided by the element indexed as n

{1,3,5,7}, and for node 5: {3,4,6,7}.

This first shallow analysis works to yield the intuition that the behaviour of an isolated node can be expressed as a series of regularities in terms of its inputs. In this context, intuition tells us that the greater the number of regularities, the shorter the description; then if no patterns are detected the chances of a causal relationship are lower.

Perspective changes when rule/algorithm or compressed expression of behaviour is not constructed from regularities identified at first sight, but from intrinsic algorithmic properties. In this latter case, behaviour of systems can be expressed as patterns of information with a distribution replicable at different scales, what we here call *fractal representation* or *fractal behaviour*. To explain what we mean by fractal, we introduce characteristics of distribution of information for the 7-node system shown in Figure 6 analyzed using ϕ_K. This implementation is shown in Table 3.

Table 3 shows how behaviour of the system shown in Figure 6 can be expressed as simple rules following an analysis based on a querying scheme that results in a reduced form to express its information distribution as a pattern replicated at different scales or as a fractal form. Answers given by systems join facts explored above on regularities and the fractal distribution of information. It is important to note that the querying scheme has to be computable and algorithmically random in order to avoid introducing an artificially random-looking behaviour from the observer (experimenter/interrogator) to the observed (the system in question).

In Table 3, after defining the target system by means of an adjacency matrix and a dynamics vector (lines 1 to 8), ϕ_K can be regarded as testing: *how 0 is distributed in node 4 in the system of seven nodes* (line 10).

The target system reacts to the ϕ_K's query and it "answers" in a compressed form (Table 3, second part, lines 1 and 2). The result can be represented in compressed form, expressed as a tiny rule that represents what we have called a fractal pattern. Such an expression is defined, as can be seen in the second square

```
1  cm07 = {{0, 0, 1, 0, 0, 0, 1},
2  {0, 0, 1, 0, 0, 1, 0},
3  {1, 0, 0, 0, 1, 0, 1},
4  {1, 0, 1, 0, 1, 0, 1},
5  {0, 0, 1, 1, 0, 1, 1},
6  {1, 1, 1, 0, 0, 0, 0},
7  {0, 1, 0, 1, 1, 1, 0}};
8  dyn07 = {"AND", "OR", "OR", "AND", "OR", "OR", "AND"};
9  (*computing places in output repertoire where node 4 = 0*)
10 res070=onPossibleBehaviour[{4}, {0}, dyn07, cm07]
11 (*Summarized representation of fractal behaviour*)
12 gp=givePlaces[res070["DecimalRepertoire"],res070["Sumandos"]];
```

```
1  <|"DecimalRepertoire"-> {0, 1, 4, 5, 16, 17, 20, 21, 64, 65,
       68, 69,80, 81, 84},
2  "Sumandos"-> {0, 2, 8, 10, 32, 34, 40, 42}|>
```

```
1  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
       32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
       46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
       60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
       74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 86, 88, 89,
       90, 91, 92, 94, 96, 97, 98, 99, 100, 101, 102, 103, 104,
       105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
       116, 118, 120, 121, 122, 123, 124, 126}
```

Table 3: ϕ_K asking for accounts of information distribution in behaviour of 4th node of the system shown in Figure 6. **Lines 1-8**: Definition of the 7-node system by means of adjacency matrix and its internal dynamics. **Line 10:** ϕ_K's code asking for zero digit location in the whole behaviour of node 4. **Line 12:** Compressing answer given by the system in line 10. **Lines 1 and 2 in results square:** Compressed form of the 0 digit distribution in the behaviour of node 4. The second grey square above shows the unfolded answer of the system.

in Table 3, by two variables: *DecimalRepertoire* that holds points distanced in different proportions where the patterns defined by the *Sumandos* variable must be reproduced. This means that in order to unfold the whole distribution (of digit zero), the pattern of numbers in *Sumandos* must be added to each value in *DecimalRepertoire*.

Once this 'fractal' simple rule is unfolded, we obtain the ordinal places where, in the whole behaviour of node 4, digit 0 can be found (see third square in Table 3). The accuracy of this answer can be verified by counting ordinals where, for node 4, its output = 0 in Table 6 in Appendix A, taking into account that counting starts at 0.

In summary, ϕ_K is turned into a kind of interrogator that asks a system about its own behaviour. On the other hand, a system is implemented as a set of rules that answers in different ways, depending on the information requested. This is unlikely with traditional approaches to ϕ, whose representation of the system consists of the whole output repertoire of the system, which might represent an important disadvantage when large networks are analyzed. ϕ_K's answers use compressed forms taking advantage of the fractal distribution of the information in the behaviour of the system, for which the answering interface is a function of its input related to each node in question.

Obviously the whole behaviour of a system is not about isolated elements, but about elements interacting in a non-linear manner, as IIT 3.0 makes clear. This last, broader view is also addressed in terms of ϕ_K, and explained in the following sections. In the next one, the advantages of simple rules over classical/naive approaches based on an exhaustive calculus and review of whole repertoires held in memory will be established.

4.3.1 Automatic meta-perturbation test

It can be seen that this querying system is similar to the programmability tests suggested in [35, 36, 18] based on questions designed to ascertain the programmability of a dynamical system.

The last section shows that systems implemented as simple rules that give rise to complex behaviour enable the system itself to "respond" to questions about where, in the chain of digits that conform to its behaviour (of a specific node), a certain pattern is to be found. And the fractal nature of information distribution in behaviour allows us to answer complex distribution questions in short forms.

In this section we show the advantages of using an (automatic) perturbation test based on simulation of behaviour using simple rules over the original version of IIT 3.0 based on the "bottleneck principle" [27] in computing integrated information.

Taking up the original perturbation test, questions take the form: *what is the output (answer) given this query (input)?*. But in ϕ_K, since questions look for explanations of the behaviour of the system itself, they take the form: *tell me if this pattern is reachable, and if so, tell me where, in the behavioural map, it is possible to find it.*
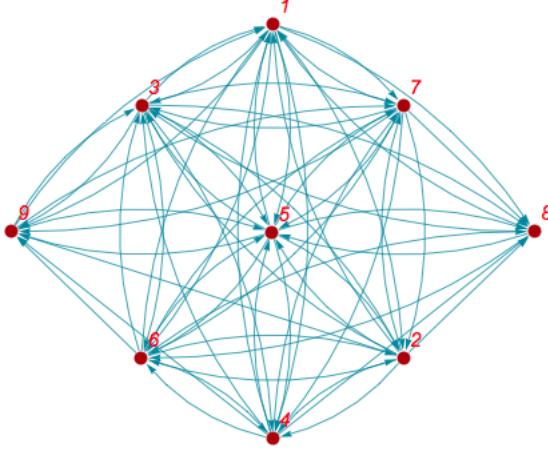
Figure 7: Network example with 9 nodes

An example of how ϕ_K implementations turn into an automatic interrogator is shown in Table 4, which aims to analyze the system networks shown in Figure 8.

In line 2, in the list code shown in Table 4, it is possible to see how ϕ_K asks questions of a system. This line should be interpreted as, *Can you compute the pattern {8,9} = {1,1} when {8,9}-> {"OR", "AND"}? If yes, tell me under what conditions you can do so.*

In this example, in the first place ϕ_K tries to find conditions needed to compute a specific output. As Table 4 shows, the answer is: *Yes! I can. This may happen when {1,3,5,6,7} = {{1,1,1,0,1},{1,1,1,1,1}}.* In this answer {1,3,5,6,7} is the set of inputs to the subsystem {8,9}.

The reader would note here that the answers offered by the system actually are conditions or inputs needed by the system to compute specific input in a format equivalent to Holland's schemas. The schemas' equivalent form for this case would be: {{1,*,1,*,1,0,1,*,*}, {1,*,1,*,1,1,1,*,*}}, where '*' is a wildcard that means 0/1 (any symbol). Such schemas correspond exactly to the generalized answer offered by the system, that is: {1,3,5,6,7} = {{1,1,1,0,1},{1,1,1,1,1}}.

This answer, like the Holland's schema theorem [20], works by imitating genetics, where a set of genes are responsible for specific features in phenotypes. What ϕ_K retrieves is the general information that yields specific inputs for the current system.

Probably the greatest advantage of the approach using ϕ_K in querying samples has to do with the computation time needed to retrieve such information, as compared with a traditional (brute force) approach: 1/10 in this case. (For results of brute force calculation see Appendix, Table 7).

This last suffices as proof that compression and generalization of systems in the form of simple rules based on naturally fractal information distribution

```
1  mmu=MemoryInUse[];
2  findPatternInSharedInputs[{1, 3, 5, 6}, "OR", 1, {1, 5, 7, 3},
        "AND", 1]//AbsoluteTiming
3  MemoryInUse[] −mmu
```

```
1  {{1, 1, 1, 0, 1}, {1, 1, 1, 1, 1}}}
2  0.000736
3  2440
```

Table 4: ϕ_K algorithmic querying of the system about its own behaviour as shown in Figure 8. **Line 2**: Query: Is it possible for this system to compute *{8,9} = {1,1} when {8,9}-> {"OR", "AND"} and whose input nodes are {1,3,5,6} and {1,5,7,3} respectively?.* The results show that the system does compute it when {1,3,5,6,7} = {{1,1,1,0,1},{1,1,1,1,1}}

has advantages over common sense or classical approaches to the analysis of complex systems, particularly in terms of the computational resources needed to compute integrated information.

All the above were applied to analyzing isolated or very simple cases. In the next section the generalization of n nodes of the system is addressed, and how this works to compute integrated information according to IIT.

4.3.2 Shrinking after dividing to rule

In previous sections it was shown how ϕ_K, applying a perturbation test, can deduce, firstly, what a system is capable of computing and the conditions under which a computation could be performed, and secondly, that by means of simple rules specifying a system it is possible to obtain descriptions of its behaviour in the form of rules that say how information is distributed, or in other words, where, in ordinal terms, such conditions can be found.

The ultimate objective of obtaining this kind of description of the behaviour of a system is to know how many times specific patterns appear in whole repertoires, and thus to construct probability distributions without need of exorbitant computational resources, since these probability distributions are a key piece used by IIT to compute integrated information.

ϕ_K addressed such challenges using a two-pronged strategy consisting firstly of parallelizing the analytical process– which is no more than a technical strategy available to be implemented in almost any computer language and that falls beyond the scope of this paper–and secondly of the partition of the target sets. This latter part of ϕ_K's strategy consists of two parts: 1) given a target set to be analyzed, to divide this into parts to be interrogated by ϕ_K via the implementation of an automatic test, and 2) to find the MIP or the Maximal Information Partition using the algorithm proposed and proved by Oizumi in [23].

In the context of ϕ_K, when a partition of a subject system is being analyzed,

the search space for the remaining parts is significantly reduced, facilitating and accelerating the analysis of the remaining parts.

In order to illustrate this idea, take for example Table 5.

Table 5 shows the definition of a system of 7 nodes (lines 1-9), where a set of a progressively growing length is searched (lines 10-14). In this example ϕ_K repeatedly asks the system if it is capable of finding a growing pattern of zeros. If it is, the system is requested to show where it is possible to find the desired pattern. Obviously, larger patterns need more computations, but as can be seen in Table 5, in the results square, the time used by ϕ_K increases as the pattern's length increases (Table 5, results square, lines 1-5), but it grows linearly in contrast to IIT 3.0, where it grows exponentially.

4.4 Limitations

When we visualize the behaviour of a system (or subsystem like an isolated node), and take into account its implementation, from the point of view of optimization of computational resources, running rules to generate the behaviour of the whole is still a challenge because it is an expensive process in terms of time and memory. Hence for large systems, analysis based on exhaustive reviews of such behaviour could eventually become intractable.

In order to overcome this limitation, ϕ_K attempted to find rules that not only give an account of the computability capabilities of a system, but also describe its own behaviour. In other words, we wanted to know about possibilities for finding "shortcuts to express the behaviour" of a whole system.

One other obvious limitation inherited from computability and algorithmic complexity is that of the semi-computability of the process of trying to find simple representations of behaviour. However, we are not required to find the shortest (simplest) one but simply a set of possible short (simple) ones, which would be an indication of the kind of system we are dealing with. While one can find shorter descriptions using popular lossless compression algorithms, the more powerful the algorithms to find shortcuts and fractal descriptions, the faster the computation and the more telling the results, something that is to be expected for a relationship between the way in which integrated information is estimated, on the one hand, and algorithmic complexity.

```
1  cm07 = {
2  {0, 0, 1, 0, 0, 0, 1},
3  {0, 0, 1, 0, 0, 1, 0},
4  {1, 0, 0, 0, 1, 0, 1},
5  {1, 0, 1, 0, 1, 0, 1},
6  {0, 0, 1, 1, 0, 1, 1},
7  {1, 1, 1, 0, 0, 0, 0},
8  {0, 1, 0, 1, 1, 1, 0}};
9  dyn07 = {"AND", "OR", "OR", "AND", "OR", "OR", "AND"};
10 onPossibleBehaviour[{1,2,3},{0,0,0},dyn07,cm07]//
      AbsoluteTiming
11 onPossibleBehaviour[{2,3,4},{0,0,0},dyn07,cm07]//
      AbsoluteTiming
12 onPossibleBehaviour[{1,2,3,4},{0,0,0,0},dyn07,cm07]//
      AbsoluteTiming
13 onPossibleBehaviour[{5,6,7},{0,0,0},dyn07,cm07]//
      AbsoluteTiming
14 onPossibleBehaviour[{1,2,3,4,5,6,7},{0,0,0,0,0,0,0},dyn07,cm07
      ]//AbsoluteTiming
```

```
1  {0.00076,<|"DecimalRepertoire"->{0},"Sumandos"->{0,2,8,10}|>}
2  {0.000647,<|"DecimalRepertoire"->{0},"Sumandos"->{0,2,8,10}|>}
3  {0.0009,<|"DecimalRepertoire"->{0},"Sumandos"->{0,2,8,10}|>}
4  {0.000656,<|"DecimalRepertoire"->{0,16},"Sumandos"->{0}|>}
5  {0.001248,<|"DecimalRepertoire"->{0},"Sumandos"->{0}|>}
```

Table 5: Comparing processing time when a system is divided to compute outputs. **Line 10-14:** ϕ_K asking the system defined in lines 1-9 for patterns filled with zeros with different lengths (3, 4 and 7) and combinations. **Lines 1-5** in results square shown, time in seconds taken for computations and answers in terms of indexes using compressed notation. In first data of results square it can be observed that the larger the node wanted, the greater the amount of time required to perform the computation, while the time ratio decreases.

5   Conclusions and future directions

Here we have established connections and developed a technique to estimate integrated information using estimations of algorithmic information, which in turn has a solid mathematical foundation. Our computational procedure has targeted what is referred to as the IIT 3.0 version, defined as a calculus of probability distributions. Instead of considering distances between statistical distributions, we formulate the problem as a distance in an algorithmic complexity space, properly approximated, in response to perturbations of the system.

Interestingly, such a perturbation programmability test–initially inspired by the Turing test (establishing another interesting connection between these new theories of consciousness and old ones)– as applied to physical systems, is a working strategy to find explanations for the behaviour of systems. It remains for future work to make conceptual and computational connections to what Oizumi and Tononi et al. called the MIP (Minimum Information Partition) [27] of a system. Having this first version of ϕ_K, we conjecture that MIP definitions also obey and are connected to algorithmic complexity in about the same way, as they should remain based on rules of an algorithmic nature. Thus, the next step is to go further in the application of the test introduced in this paper to discover simple rules that would help to find MIP in a more natural and a faster way. Another possible direction is to systematize the finding of these simple rules and apply more powerful methods to enable computation of larger systems. However, here we have merely established the first principles and the directions that can be explored following these ideas.

Finally, we think that these ideas about self-explanatory systems capable of providing answers to questions about their own behaviour can help in devising techniques to make other methods, in areas such as machine and deep learning, explain their own, often obscure, behaviour.

References

[1] D.H. Bailey, P.B. Borwein, S. Plouffe, On the Rapid Computation of Various Polylogarithmic Constants, *Math. Comput.* 66, 903-913, 1997.

[2] Crutchfield, J.P., Shalizi, C.R Thermodynamic depth of causal states: Objective complexity via minimal representations. *Journal of statistical physics, Springer*, E 1, 59, 275–283, 1999.

[3] Shalizi, Cosma Rohilla and Crutchfield, James P. Computational mechanics: Pattern and prediction, structure and simplicity. *Journal of statistical physics, Springer*, 104(1):817–879, 2001.

[4] Shalizi, C.R., Shalizi, K.L. Optimal nonlinear prediction of random fields on networks. *Discrete Math Theor Comput Sci, AB(DMCS)*, 11–30, 2003.

[5] T. Schreiber, Measuring Information Transfer, *Physical Review Letters*, 85 (2): 461–464, 2000.

[6] R.J. Solomonoff, A formal theory of inductive inference: Parts 1 and 2.

[7] G. Tononi, M. Boly, M. Massimini, C. Koch. "Integrated information theory: from consciousness to its physical substrate". Nature Reviews Neuroscience. 17 (7): 450–461, 2016.

[8] C. Villani. Topics in Optimal Transportation, *American Mathematical Society*, Providence, RI, 2003.

[9] Devine, S., The Application of Algorithmic InformationTheory to Noisy Patterned Strings . *Complexity*, 12(2), 52-58, 2006

[10] M. Rosanova M. Boly S. Sarasso K.R. Casali S. Casarotto M.-A. Bruno S. Laureys G. Tononi A.G. Casali, O. Gosseries and M. Massimini. A theoretically based index of consciousness independent of sensory processing and behavior. *Science Translation Medicine*, 5(198):198ra105, 2013.

[11] P. W. Anderson. More is different. *science*, 177(4047):393–396, 1972.

[12] Albert-Laszlo Barabasi and Márton Posfai. *Network Science)*. Oxford University Press, 2016.

[13] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017.

[14] G. J Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM (JACM)*, 13(4):547–569, 1966.

[15] David G Chandler. *introduction to Modern Statistical Mechanics)*. Oxford University Press, 1987.

[16] Peter Dayan and L.F. Abbott. *Theoretical NeuroscienceComputational and Mathematical Modeling of Neural Systems*. MIT Press, 2001.

[17] 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*, 219(1):63–77, 2012.

[18] F. Marabita Y. Deng S. Elias A. Schmidt G. Ball J. Tegnér H Zenil, N.A. Kiani. An algorithmic information calculus for causal discovery and reprogramming systems. *iScience*, (in press).

[19] C.W.J. Granger, Investigating causal relations by econometric models and cross-spectral methods, *Econometrica* 37, 424–438, 1969.

[20] J. H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. U Michigan Press, 1975.

30

[21] S.H. Strogatz, *Nonlinear Dynamics And Chaos: With Applications To Physics, Biology, Chemistry, And Engineering (Studies in Nonlinearity)*. CRC Press, 2000.

[22] V. Iapascurta. Detection of movement towards randomness by applying block decomposition method to a simple model of circulatory system. *Complex Systems*, 28(1), 2019.

[23] J. Kitazono, R. Kanai, and M. Oizumi. Efficient Algorithms for Searching the Minimum Information Partition in Integrated Information Theory. *Entropy*, 20(3):173, March 2018. arXiv: 1712.06745.

[24] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information and Transmission*, 1(1):1–7, 1965.

[25] W.G.P. Mayner, W. Marshall, L. Albantakis, Graham Findlay, R. Marchman, and G. Tononi. PyPhi: A toolbox for integrated information theory. *arXiv:1712.09644 [cs, q-bio]*, December 2017. arXiv: 1712.09644.

[26] M. Oizumi, L. Albantakis, and G. Tononi. From the phenomenology to the mechanisms of consciousness: integrated information theory 3.0. *PLoS Comput Biol*, 10(5):e1003588, 2014.

[27] M. Oizumi, L. Albantakis, and G. Tononi. From the phenomenology to the mechanisms of consciousness: integrated information theory 3.0. *PLoS Comput Biol*, 10(5):e1003588, 2014.

[28] J. Pearl, *Causality: Models, Reasoning, and Inference*, Cambridge University Press, Cambridge, UK, 2000.

[29] G. Ruffini. An algorithmic information theory of consciousness. *Neuroscience of Consciousness*, 1(nix019), 2017.

[30] F. Soler-Toscano, H. Zenil, J.-P. Delahaye, and N. Gauvrit. Calculating kolmogorov complexity from the output frequency distributions of small turing machines. *PLoS ONE*, 9(5):1–18, 2014.

[31] M. Ventresca. Using algorithmic complexity to differentiate cognitive states in fmri. *In: Aiello L., Cherifi C., Cherifi H., Lambiotte R., Lió P., Rocha L. (eds), Complex Networks and Their Applications VII. COMPLEX NETWORKS 2018. Studies in Computational Intelligence, vol 813, Springer*, 2018.

[32] P.L. Williams, R.D. Beer, Nonnegative decomposition of multivariate information, arXiv:1004.2515, 2010

[33] P.L. Williams, R.D. Beer, Generalized measures of information transfer, arXiv:1102.1507, 2011.

[34] H. Zenil. A behavioural foundation for natural computing and a programmability test. In *Computing Nature*, pages 87–113. Springer, 2013.

[35] H. Zenil. A behavioural foundation for natural computing and a programmability test. In G. Dodig-Crnkovic and R. Giovagnoli, editors, *Computing Nature: Turing Centenary Perspective*, volume 7, pages 87–113. Springer SAPERE, 2013.

[36] H. Zenil. Testing biological models for non-linear sensitivity with a programmability test. In G. Nicosia S. Nolfi P. Lió, O. Miglino and M. Pavone, editors, *Advances in Artificial Intelligence*, pages 1222–1223. MIT Press, 2013.

[37] H. Zenil. Algorithmic data analytics, small data matters and correlation versus causation. *In M. Ott, W. Pietsch, J. Wernecke (eds.), Berechenbarkeit der Welt? Philosophie und Wissenschaft im Zeitalter von Big Data (Computability of the World? Philosophy and Science in the Age of Big Data) Springer Verlag*, pages 453–475, 2017.

[38] H. Zenil, L. Badillo, S. Hernández-Orozco and F. Hernández-Quiroz. Coding-theorem Like Behaviour and Emergence of the Universal Distribution from Resource-bounded Algorithmic Probability *International Journal of Parallel Emergent and Distributed Systems*, 2018.

[39] H. Zenil, S. Hernández-Orozco, N.A. Kiani, F. Soler-Toscano, A Rueda-Toicen, and J. Tegnér. A decomposition method for global evaluation of shannon entropy and local estimations of algorithmic complexity. *Entropy*, 20(8):605, 2018.

[40] H. Zenil, N.A. Kiani, and J. Tegnér. Methods of information theory and algorithmic complexity for network biology. *Seminars in Cell and Developmental Biology*, 51:32–43, 2016.

[41] H. Zenil, N.A. Kiani, and J. Tegnér. Low algorithmic complexity entropy-deceiving graphs. *Physical Review E*, 96(012308), 2017.

[42] H. Zenil, N.A. Kiani, and J. Tegnér. A review of graph and network complexity from an algorithmic information perspective. *Entropy*, 20(8):551, 2018.

[43] H. Zenil, N.A. Kiani, and J. Tegnér. Algorithmic information dynamics of emergent, persistent, and colliding particles in the game of life in a. adamatzky (ed) pp.367–383. *From Parallel to Emergent Computing, Taylor & Francis / CRC Press*, 2019.

[44] H. Zenil, N. A. Kiani, and J. Tegnér. *Algorithmic Information Dynamics: A Computational Approach to Causality and Molecular Biology. From Networks to Cells.* Cambridge University press, (forthcoming).

[45] H. Zenil, A. Schmidt, and J. Tegnér, Causality, information and biological computation: An algorithmic software approach to life, disease and the

immune system. In P.C.W. Davies S.I. Walker and G. Ellis, editors, *Information and Causality: From Matter to Life*, pages 244–279. Cambridge University Press, 2017.

[46] H. Zenil, F. Soler-Toscano, K. Dingle, and A. Louis. Correlation of automorphism group size and topological properties with program-size complexity evaluations of graphs and complex networks. *Physica A: Statistical Mechanics and its Applications*, 404:341–358, 2014.

A Appendix

How the Meta-perturbation test works

In order to explain the advantages of the generalization of information in the form of schemas computed by simple rules, Table 7 is introduced. In this Table are shown all possible cases where the pattern {8,9}={1,1} can be found in the whole output repertoire of the system introduced in Figure 8 in the main text.

On the right side of the set contained in Table 7, outputs where {8,9}={1,1} are highlighted in red. On the left side the 9-length are inputs that yield to outputs containing the desired pattern. On this left side, in bold, are the corresponding inputs that are particularly responsible for causing the desired pattern, that is, all possible patterns for the inputs {1,3,5,6,7}

In order to obtain the results in Table 8 using a naive (brute force) approach, it was necessary to define the whole set of all 2^9 possible inputs and compute the whole set of outputs; then an exhaustive search for {8,9}={1,1} was carried out. Notice that time and memory used are at least 10 times greater than those used in the ϕ_K approach. These results are shown in the last two rows in Table 8 and the results square in Table 4.

```
cm07 = {{0, 0, 1, 0, 0, 0, 1},
{0, 0, 1, 0, 0, 1, 0},
{1, 0, 0, 0, 1, 0, 1},
{1, 0, 1, 0, 1, 0, 1},
{0, 0, 1, 1, 0, 1, 1},
{1, 1, 1, 0, 0, 0, 0},
{0, 1, 0, 1, 1, 1, 0}};
dyn07 = {"AND", "OR", "OR", "AND", "OR", "OR", "AND"};
analysis07=runDynamic[cm07, dyn07]["RepertoireOutputs"]
```

```
{{0, 0, 0, 0, 0, 0, 0}, {0, 0, 1, 0, 0, 1, 0}, {0, 0, 0, 0, 0, 1, 0}, {0, 0, 1, 0, 0, 1, 0},
{0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 0, 0, 0, 1, 0, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 0, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0},
{0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 0, 1, 0, 0, 0, 0}, {0, 0, 1, 0, 0, 1, 0}, {0, 0, 1, 0, 0, 1, 0}, {0, 0, 1, 0, 0, 1, 0},
{0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 0, 1, 0, 1, 0, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0},
{0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 1, 0, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 1, 0, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 0, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 1, 1, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{0, 1, 1, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 1}, {0, 1, 1, 0, 1, 1, 1},
{0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 1}, {0, 1, 1, 0, 1, 1, 1},
{0, 0, 1, 0, 1, 0, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0},
{1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0},
{0, 0, 1, 0, 1, 0, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0},
{1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0},
{0, 0, 1, 0, 1, 0, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0},
{1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 1, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 1, 1, 1, 0},
{0, 0, 1, 0, 1, 0, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0}, {0, 0, 1, 0, 1, 1, 0},
{1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 1, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 1, 1, 1, 0},
{0, 1, 1, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0},
{0, 1, 1, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0},
{0, 1, 1, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 0},
{1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 1, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 1, 1, 1, 0},
{0, 1, 1, 0, 1, 0, 0}, {0, 1, 1, 0, 1, 1, 0}, {0, 1, 1, 0, 1, 1, 1}, {0, 1, 1, 0, 1, 1, 1},
{1, 1, 1, 0, 1, 1, 0}, {1, 1, 1, 1, 1, 1, 0}, {1, 1, 1, 0, 1, 1, 1}, {1, 1, 1, 1, 1, 1, 1}}
```

Table 6: Seven-node system and output repertoires. (A) Network definition by adjacency matrix (lines 1-7) and dynamics (line 8). (B) Output repertoire.

```
1  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1}
```

```
1  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
      18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
      32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
      46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
      60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
      74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 86, 88, 89,
      90, 91, 92, 94, 96, 97, 98, 99, 100, 101, 102, 103, 104,
      105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
      116, 118, 120, 121, 122, 123, 124, 126}
```

Table 7: Comparison between real behaviour and unfolded nested rule of behaviour of node 4 of the system defined in Figure 6 of the main text. The answer offered by the system (second results square) shows the places where digit 0 is found in the behaviour of node 4 (first results square)

{**1**,0,**1**,0,**1**,**0**,**1**,0,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,0,**1**,**0**,**1**,0,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,1,**1**,**0**,**1**,0,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,1,**1**,**0**,**1**,0,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,0,**1**,**1**,**1**,0,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,0,**1**,**1**,**1**,0,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,1,**1**,**1**,**1**,0,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,1,**1**,**1**,**1**,0,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,0,**1**,**0**,**1**,1,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,0,**1**,**0**,**1**,1,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,1,**1**,**0**,**1**,1,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,1,**1**,**0**,**1**,1,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,0,**1**,**1**,**1**,1,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,0,**1**,**1**,**1**,1,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,1,**1**,**1**,**1**,1,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,1,**1**,**1**,**1**,1,0}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,0,**1**,**0**,**1**,0,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,0,**1**,**0**,**1**,0,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,1,**1**,**0**,**1**,0,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,1,**1**,**0**,**1**,0,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,0,**1**,**1**,**1**,0,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,0,**1**,**1**,**1**,0,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,1,**1**,**1**,**1**,0,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,1,**1**,**1**,**1**,0,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,0,**1**,**0**,**1**,1,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,0,**1**,**0**,**1**,1,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,0,**1**,1,**1**,**0**,**1**,1,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,1,**1**,**0**,**1**,1,1}->{1,1,0,1,0,1,1,**1**,**1**},
{**1**,0,**1**,0,**1**,**1**,**1**,1,1}->{1,1,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,0,**1**,**1**,**1**,1,1}->{1,1,0,0,0,0,1,**1**,**1**},
{**1**,0,**1**,1,**1**,**1**,**1**,1,1}->{1,0,0,1,0,0,1,**1**,**1**},
{**1**,1,**1**,1,**1**,**1**,**1**,1,1}->{1,0,1,0,1,1,1,**1**,**1**},
0.166334
217920

Table 8: Naive approach to looking for algorithmic patterns based on simplicity vs complexity in the calculation of integrated information.

B Appendix

Algorithm 1: computeIntegratedInformation
---
    input : AdjacencyMatrix,Dynamic,CurrentState
    output: Information Integration Value

1 nodes $\leftarrow$ GetNodes($AdjacencyMatrix$);
    // UPPD: Unrestricted Past Probability Distribution;
    // UFPD: Unrestricted Future Probability Distribution;
2 UPPD $\leftarrow$
    ComputesPastProbabilityDistribution(nodes,$CurrentState$,$\emptyset$,$Dynamic$,am);

3 UFPD $\leftarrow$
    ComputesFutureProbabilityDistribution(nodes,$CurrentState$,$\emptyset$,$Dynamic$,am);

    // am=AdjacencyMatrix; cs=CurrentState;
4 conceptualSpace $\leftarrow$
    ComputesConceptualSpace(am,$Dynamic$,cs,UPPD,UFPD);
5 integratedInformationValue $\leftarrow 0$;
6 bipartitionsSet $\leftarrow$ Bipartitions(conceptualSpace);
7 **foreach** $bipartition$ b_i $\in$ bipartitionsSet **do**
8   aux $\leftarrow$EMD(b_i,conceptualSpace);
9   **if** aux $>$ integratedInformationValue **then**
10   integratedInformationValue $\leftarrow$ aux;

Algorithm 2: computeConceptualSpace
---
    input : AdjacencyMatrix,Dynamic,CurrentState,UPPD,UFPD
    output: conceptualStructure

    // UPPD: Unrestricted Past Probability Distribution;
    // UFPD: Unrestricted Future Probability Distribution;
1 nodes $\leftarrow$ GetNodes($AdjacencyMatrix$);
2 mechaSet $\leftarrow$Subsets(nodes);
3 **foreach** $mechanism$ $mecha_i$ $\in$ mechaSet **do**
4   OneConcept
        $\leftarrow$ComputeConceptOfAMechanism($mecha_i$,nodes,$CurrentState$,$UPPD$,$UFPD$);

5   Append(conceptualSpace,OneConcept)

Algorithm 3: computeConceptOfAMechanism

---

    input : mechanism,nodesForPurviews,currentState,pastDistro,futDistro,Dynamic, AdjacencyMatrix

    output: concept for current mechanism

    `// nodes where all purviews will be taken from`

1 purviewsSet ←Subsets($nodesForPurviews$);

2 **for** $j \leftarrow 1$ **to** Length($PurviewsSet$) **do**

3   aPurview ← Part($j, PurviewsSet$);

4   connected ← FullyConnectedQ($mechanism, APurview$);

5   **if** connected **then**

        `// MIP: Maximal Information Partition`

6   smallAlpha ← ComputesMIP($mechanism, APurview$) ;

    `// APurviewMIP: Purview responsable to cause MIP for`
    `current mechanism;`

7 aPurviewMIP ← smallAlpha("$PurviewMIP$");

    `// Following sum is formalized in Figure 4, In Text S2`
    `from Oizumi(2014);`

    `// cs=CurrentState; am = AdjacencyMatrix;`

8 pastDistribution ←
    ComputesPastProbabilityDistribution($mechanism, cs$, aPurviewMIP,
    $Dynamic, am$);

9 futureDistribution ←
    ComputesFutureProbabilityDistribution($mechanism, cs$, aPurviewMIP,
    $Dynamic, am$);

10 ConceptualInfo ←
    EMD($pastDistro$, pastDistribution)+EMD($futDistro$, futureDistribution);

Algorithm 4: ComputesMIP
---

    input : mechanism,purview
    output: MIP structure

1 mechaChildren ← Subsets($mechanism$);
2 PurviewChildren ← Subsets($purview$);
3 ci ← 10000;
4 ei ← 10000;
5 **foreach** $mecha$ m_i ∈ mechaChildren **do**
6   **foreach** $purview$ p_i ∈ PurviewChildren **do**
            // cs=CurrentState, am=AdjacencyMatrix;
7   pastDistribution ←
          ComputesPastProbabilityDistribution(m_i,cs,p_i,$Dynamic$,am);

8   futureDistribution ←
          ComputesFutureProbabilityDistribution(m_i,cs,p_i,$Dynamic$,am);

9   cei ←
          ComputesCEI($mecha$,$purview$,pastDistribution,futureDistribution);

10   **if** cei *("ci")* < ci **then**
11   ci ← cei ("ci");
12   pastMIP ← (mecha,purview);
13   **if** cei *("ei")* < ei **then**
14   ei ← cei ("ei");
15   futMIP ← (mecha,purview);

Algorithm 5: computesCEI

---

input : ChildMecha, ChildPurview, ParentMecha, ParentPurview,
ParentPastDistro, ParentFutDistro,
UnconstrainedPastDistro, UnconstrainedFutDistro

output: Causal and Effect information values

---

1 mechaComplement ← ComplementI($ChildMecha, ParentMecha$)

2 purviewComplement ←
ComplementI($ChildPurview, ParentPurview$)

3 mechaChildren ← Subsets($mechanism$)

4 PurviewChildren ← Subsets($purview$)

5 **foreach** $mecha\ m_i \in$ mechaChildren **do**

6   **foreach** $purview\ p_i \in$ PurviewChildren **do**

7   ChildMecha ← m_i

8   ChildPurview ← p_i

9   **if** *ChildMecha*=∅ **then**

10   pastDistribution ← UnconstrainedPastDistro

11   futureDistribution ← UnconstrainedFutDistro

12   **else if** *ChildPurview*=∅ **then**

13   pastDistribution ← 1

14   futureDistribution ← 1

15   **else if then**

16   pastDistribution ←
ComputesPastProbabilityDistribution($ChildMecha, ChildPurview, cm, am$)

17   futureDistribution ←
ComputesFutureProbabilityDistribution($ChildMecha, ChildPurview, cm, am$)

18

19   **if** mechaComplement =∅ **then**

20   pastDistributionComp ← UnconstrainedPastDistro

21   futureDistributionComp ← UnconstrainedFutDistro

22   **else if** purviewComplement =∅ **then**

23   pastDistributionComp ← 1

24   futureDistributionComp ← 1

25   **else if then**

                   // CPPD=ComputesPastProbabilityDistribution

                   // CFPD=ComputesFutureProbabilityDistribution

26   pastDistributionComp ←
CPPD($MechaComplement, PurviewComplement, cm, am$)

27   futureDistributionComp ←
CFPD($MechaComplement, PurviewComplement, cm, am$)

28   pastDistribution ← Normalize(pastDistribution
*pastDistributionComp)

29   futureDistribution ← Normalize(futureDistribution
*futureDistributionComp)

30   ci ← EMD($ParentPastDistro$,pastDistribution)

31   ei ← EMD($ParentFutDistro$,futureDistribution)

32   cei ← Min(ci,ei)

41

Algorithm 6: computesPositionsOfAPatternInOutputs

---

    input : Mechanism,Purview,AdjacencyMatrix,CurrentState,Dynamic

    output: positions (indexes) where current state of mechanism is
               found into the output repertoire

```
// work as nodes that send inputs to the mechanism.
// This remaning nodes are actually powers that define a
pattern
// of distribution of the wanted pattern defined by
mechanism
```

1 joinedNames ← Join(Inputs($Mechanism$))

2 powers ← Complement(Range(Length($AdjacencyMatrix$)),
  joinedNames)-1

3 foreach $node\ n_i \in$ powers **do**

4 | Append(sumandos, 2^{n_i})

5 sumandos ←Subsets(sumandos)

6 foreach $sumando\ s_i \in$ sumandos **do**

7 | Append(Aux, Sum(s_i))

8 sumandos ←Aux

9 ins ← Inputs(FirstNode($Mechanism$))

```
// Given a dynamic it computes all possible inputs of
defined size that
// results in a defined output (cs)
```

10 repertoire ← RepertoireByOutput(Length(ins),$Dynamic$,cs)

11 foreach $node\ m_i \in$ (mechanism − FirstNode(mechanism)) **do**

12 | intersec ← Intersection(ins,Inputs(m_i))

13 | ins ← ins +(Inputs(m_i)-intersec)

14 | repertoire
        ←Combine(repertoire,$CreateRepertoire($ Inputs(m_i)-intersec$)$)

15 | repertoire ←FilterRepertoireByOutput(repertoire,cs)

16 foreach $sumando\ s_i \in$ sumandos **do**

17 | **foreach** $repert\ r_i \in$ repertoire **do**

18 | | Append(indexes, Sum(s_i, r_i))

Algorithm 7: computesPastProbabilityDistribution

input : Mechanism,Purview,AdjacencyMatrix,CurrentState,Dynamic

output: Probability distribution for a mechanism

---
1 locations
   ←computesPositionsOfAPattern($Mechanism, CurrentState, Purview, AdjacencyMatrix$)

2 allInputs ← Extract(locations,$Purview$)

3 probability ← $1/($Length($Locations$)$)$

4 **foreach** *input in_i* ∈ allInputs **do**

5 | correctedLocations
   | ←FindPatternInInputs($Purview, in_i,$ Length($AdjacencyMatrix$))

6 probabilityDistribution ←
   ComputesProbabilityForElements(correctedLocations)

Algorithm 8: computesFutureProbabilityDistribution

input : Mechanism,Purview,AdjacencyMatrix,CurrentState,Dynamic

output: Probability distribution for a mechanism

---
1 locations
   ←(FindPatternInInputs($Purview, CurrentState,$ Length($AdjacencyMatrix$)))-
   1

2 allOutputs ← ComputesOutputs(locations)

3 allInputs ← Extract(locations,$Purview$)

4 probabilityDistribution ←
   ComputesProbabilityForElements(allInputs)

Algorithm 9: findPatternInInputs

    input : Nodes,WantedPattern,sizeAdjacencyMatrix
    output: Finds indexes in input repertoire where nodes fullfill
                 wantedPattern

1   limit $\leftarrow 2^{sizeAdjacencyMatrix}$

2   **foreach** $node\ n_i \in Nodes$ **do**

3   powers $\leftarrow 2^{n_i-1}$

4   repetitions $\leftarrow$ limit/powers

5   longi $\leftarrow$ limit/repetitions

6   **if** *if expectedPatt = 1* **then**

7   serie $\leftarrow$ `CreateSerieEvenNumbers`(repetitions)

8   **else**

9   serie $\leftarrow$ `CreateSerieOddNumbers`(repetitions)

10   **for** $i=1;\ i<$ `Length`(serie)$;\ i=i+1$ **do**

11   Found $\leftarrow$
            `Range`(*((powers *serie [i])-longi)+1,* powers **serie [i]*)

Algorithm 10: computeInputBitProbabilityDistro

    input : Nodes,AdjacencyMatrix,Dynamics
    output: Bit probability for given nodes

1   locations $\leftarrow$
     `FindPatternInInputs`(*Nodes,1,*`Length`(*AdjacencyMatrix*))

2   oneProbability $\leftarrow$ `Length`(locations)$/2^{\mathbf{Length}(AdjacencyMatrix)}$

3   zeroProbability $\leftarrow 1 -$ oneProbability

Algorithm 11: computeOutputBitProbabilityDistro

    input : Nodes,AdjacencyMatrix,Dynamics
    output: Bit probability for given nodes

1   locations $\leftarrow$
     `ComputesPositionsOfAPatternInOutputs`(*Nodes,1,Dynamics,AdjacencyMatrix*)

2   oneProbability $\leftarrow$ `Length`(locations)$/2^{\mathbf{Length}(AdjacencyMatrix)}$

3   zeroProbability $\leftarrow 1 -$ oneProbability