UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"ALGORITHMIC PROBABILITY-DRIVEN AND OPEN-ENDED EVOLUTION"

T E S I S
QUE PARA OPTAR POR EL GRADO DE
DOCTOR EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
SANTIAGO HERNÁNDEZ OROZCO

TUTORES PRINCIPALES:

DR. FRANCISCO HERNÁNDEZ QUIROZ
FACULTAD DE CIENCIAS – UNAM

DR. HÉCTOR ZENIL CHÁVEZ
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN – UNAM

MIEMBROS DEL COMITÉ TUTOR:

DR. CARLOS GERSHENSON GARCÍA
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS – UNAM

CIUDAD DE MÉXICO, ABRIL 2018

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

Dedicado a mis padres
Roberto Tito Hernández López y
Susana Orozco Segovia.

Agradecimientos

Primero, me gustaría agradecer a mis padres Susana y Tito, a mis hermanos Miguel y Efraim, y a mi Tía Alma y prima Sonia por su apoyo y compañía durante todos estos años. También agradezco profundamente a Francisco Hernández Quiroz, Héctor Zenil y Carlos Gershenson García por su confianza y apoyo.

Sin ustedes este logro no hubiera sido posible.

Se agradece al Consejo Nacional de Ciencia y Tecnología y al Posgrado en Ciencia e Ingeniería de la Computación el apoyo brindado para realizar estos estudios.

Contents

Abstract

Does decidability hampers the potential of a dynamical system to generate complexity? And, given the case, under what conditions can a system exhibit non-trivial evolutionary behavior? Answering these two questions is the main task of the research presented in the current text, which is divided in two interconnected parts.

Chapters 2, 3, 4 and 5 deals with the first question within the context of computable evolutionary dynamical systems. Using methods derived from algorithmic complexity theory, we give robust definitions for open-ended evolution and the adaptability of the sate of a system. We show that decidability imposes absolute limits to the speed of complexity growth. This limit ties tightly the speed of which complexity and innovation can appear to that of the most trivial systems. Conversely, systems that exhibit non trivial open-ended evolution must be undecidable, establishing undecidability as a requirement for such systems [39].

In Chapters 6, 7, 8 we describe dynamical systems, first proposed by Chaitin, that meet our criteria for no trivial (strong) open-ended evolution, along with a series of numerical experiments on them. We call these systems algorithmic probability-driven evolutionary systems or simply algorithmic evolution. Among the experimental and theoretical results we found that regular structures are preserved and carried on when they first occur and can lead to an accelerated production of diversity and extinction, possibly explaining naturally occurring phenomena such as diversity explosions and massive extinctions. We argue that the approach introduced appears to be a better approximation to biological evolution than models based exclusively upon random uniform mutations [41, 86].

Chapter 1

Introduction

Explaining the causes and underlying mechanics of the diversity and perceived complexity found in the natural world is one of the main objectives of science, philosophy and theology. One avenue of research on the topic is found in the field known as artificial life (ALife), which embraces the use of computational tools to search for the conditions needed for evolutionary systems that are capable of exhibiting the properties of diversity and complexity that are ascribed to the natural world. It is in this field where we find the concept of Open-Ended Evolution (OEE).

1.1 The Open-Ended Evolution problem

Artificial Life (ALife), named by Christopher Langton, is a field that uses computer science to study life in general, formulating an interdisciplinary theory of mathematical biology [75]. Central to the field are the concepts of self-organization and emergence; order and structure that are thought to generate spontaneously from an initial computable set of rules or fundamental entities [60]. It is within this context that we find the concept of *Open-Ended Evolution (OEE).*

In broad terms, OEE is understood as the observed capacity that the biosphere has to create complex and innovative life forms. When applied to artificial evolution, one author describes OEE as "*evolutionary dynamics in which new, surprising, and sometimes more complex organisms and interactions continue to appear*" [73].

Giving a precise characterization for OEE and establishing the requirements and properties for a system to exhibit OEE is considered to be one of the most important open problems in the field [7, 71, 72] and is thought to be a required property of evolutionary systems capable of producing life [63]. This is called the *OEE problem.*

1.2 The Question: Complexity and Undecidability of OEE

Central to the OEE problem is the evolution of complexity within the bounds of its initial characterization. Defining and effectively measuring complexity is a problem in itself [35]. In the opinion of this author, the most successful theory of complexity is found in the framework of *Algorithmic Information Theory* (AIT) [42]. Developed independently by Chaitin [18], Solomonoff [70] and Kolmogorov [47], this theory uses the algorithmic complexity of a finite object as its foundation. This measure of complexity is invariant from the choice of (Turing compatible) computational model, therefore is *universal.*

It is within this framework that we find the following results pertaining the limits that initial conditions impose upon the rest of the system:

> "In evolutionary terms, it is difficult to justify how biological algorithmic complexity could significantly surpass that of the environment itself."
>
> Zenil et al.,[79].

> "The theorems of a finitely specified theory cannot be significantly more complex than the theory itself."

Gregory Chaitin, [19].

"Is complexity a source of Incompleteness?"

Calude and Jürgensen, [16].

For the author, the cited results and statements strongly suggest the following question:

Is undecidability a requirement for Open-Ended Evolution?

Answering this question by establishing an strict mathematical framework is the first task undertaken by the first part of this project.

1.2.1 Algorithmic Driven Evolution

The second part to this project is concerned with exploring, theoretically and experimentally, the consequences of assuming an evolutionary framework based in algorithmic probability theory. This exploration is motivated by the metabiology framework laid out by Chaitin [23, 22, 21] and assumes that mutation randomness comes from algorithmic sources.

Natural selection explains how life has evolved over millions of years from more primitive forms. However, the speed at which this happens has sometimes defied explanations based on random (uniformly distributed) mutations. If the origin of mutations is algorithmic in any form, and can thus be considered a program in software space, the emergence of a natural algorithmic probability distribution has the potential to become an accelerating mechanism.

The *central* theorem for algorithmic complexity theory establishes that the probability of an object to be generated by a randomly chosen computable process is in exponential inverse proportion to its algorithmic descriptive complexity [70]. This result can be used to approximate the algorithmic probability of an object. I simulate the application of algorithmic mutations to binary matrices using numerical approximations to algorithmic probability based on the Block Decomposition Method (BDM) [84, 86], which is a computable method for approximating the algorithmic descriptive complexity of a binary multidimensional matrix.

1.3 Contributions

The main contributions presented are:

- The development of a mathematical definition for OEE.
- A formal characterization of adaptability.
- The presentation of a mathematical proof for the undecidability of adaptation for systems that exhibit OEE.
- Exhibiting a system, first proposed by Chaitin, that show OEE.
- An argumentation of why such system is an adequate model for biological evolution.
- The development of an *in-silico* first experiment based on the concept of *algorithmic probability driven evolution.*
- The experiment based on this model showed the following behavior, which are backed by further theoretical developments:
    - Algorithmic probability driven evolution must start from random stages.
    - The presence of statistically significant speed-up when evolving towards non-trivially complex targets.
    - Detection of resilient non-trivial structures called *persistent structures.* These structure impose the following behavior:
        - Once an individual develops a regular structure during evolution, it is hard (improbable) go get rid of it.
        - This difficulty increases with the size of the space of possible mutations.

        * Therefore, persistent structures can promote both: diversity and extinctions among a population.
        * One *natural* way to mitigate the negative aspects of persistent structures is by local evolution, or modularity.
    - I argue that this behavior is a better approximation to biological evolution than models that rely on "regular" (uniform) probability for their mutations. And since algorithmic probability driven evolution converges faster to regular structures than random mutation, it approaches strong OEE behavior.

1.4 Preceding Approaches to the Topic

The relationship between dynamical systems and computability has been studied before by Bournez [10], Blondel [9], Moore [59] and by Fredkin, Margolus and Toffoli [31, 55], among others. That emergence is a consequence of incomputability has been proposed by Cooper [26]. Complexity as a source of undecidability has been observed in logic by Calude and Jürgensen [16]. Delvenne, Kurka and Blondel [30] have proposed robust definitions of computable (effective) dynamical systems and universality, generalizing Turing's halting states, while also setting forth the conditions and implications for universality and decidability and their relationship with chaos. Mechanisms and requirements for open-ended evolution in systems with a finite number of states (resource-bounded) have been studied by Adams et al. [3].

The definitions and general approach used during my research differs from the ones cited above, therefore an extensive review of the topic on such terms is not conductive to a better understanding of the topic as presented. However, the results are ultimately related.

1.5 Outline

Chapters **2** to **6** correspond to the result presented in the article *The limits of decidable states on open-ended evolution and emergence* [40], of which I was first author. These chapters contain theoretical results concerning to the concept of Open-Ended Evolution and the restrictions it imposes to the rest of the system.

Chapter **7** describes a computable method to approximate the descriptive complexity of a multidimensional object and contains excerpts from the article *A decomposition method for global evaluation of Shannon entropy and local estimations of algorithmic complexity* [86], for which I was a contributing author. For this purpose, this chapter contains a brief introduction to algorithmic complexity theory. I use this method to define a computable approximation to the algorithmic probability of a matrix.

Building upon the preceding chapters, chapter **8** describes a computable model that approaches algorithmic evolution. This model, coded in `Haskell`, was used for a series of numeric experiments. I present the obtained results, along with further theoretical developments based on them. This chapter's content overlaps with the article *Algorithmically probable mutations reproduce aspects of evolution such as convergence rate, genetic memory, modularity, diversity explosions, and mass extinction* [41], for which I was first author.

Finally, chapter **9** contains the joint conclusions of the three main sections of this thesis.

Chapter 2

Computability, Randomness and Complexity

Algorithmic information theory (AIT) is the field of mathematics and theoretical computer science that studies the information content of individual objects from an algorithmic point of view, concerning itself with the relationship between computation, information, and randomness [53, 42].

One of the problems that motivated the development of this field was finding a formal characterization for finite (and infinite) random mathematical objects. For example, consider the problem of deciding if a binary sequence of 8 bits is *truly random*, like the one expected from a fair coin toss game. Intuitively, the sequence 01010101 has low probability of being random compared to 01101001. However, traditional probability states that both have the same change of occurring: $1/2^8$. And, as AIT has found, the problem of randomness is intimately related to the problem of complexity.

One way to solve the randomness problem is to state that the sequence 01010101 was likely made by the simple *process* of repeating 01 four times, while 01101001 was likely done by *randomly* pressing keys in a keyboard. However, for this solution to be broadly applied to the problem of randomness, we need to precisely define first what is a process (or algorithm).

2.1 Computability

The Entscheidungsproblem or *decision* problem is a fundamental problem in mathematics enunciated by the influential mathematician Davil Hilbert in 1928. The problem asked for the existence of a process (algorithm) that could take any clause in a first-order logic theory and answer if it was true for any logical structure satisfying the axioms of the theory. This is equivalent to ask for a *mechanical* method for deciding if the clause can be proven (or *derived* using the rules of logic) from the initial set of axioms.

The problem was solved independently by Alonzo Church [24] and Alan Turing [74] in 1936 as a negation: no such algorithm exists. Both mathematicians described different, but equivalent, formal systems that capture *everything that can be effectively calculable or computed* following a deterministic procedure. These systems are known as λ-calculus and Turing machines, respectively. By the Church-Turing thesis, a function over the natural numbers is computable following an algorithm if and only if it is computable by a Turing machine (or a λ-expression) [27]. In other words, they formalized the notion of **algorithm**, which so far has shown to be *universal.*

For this work I will omit giving a detailed description of a Turing machine or a λ-expression. Instead I will focus in a functional approach of computability: I will consider Turing machines as functions over the natural numbers which are agreed to be *computable.*

Definition 2.1. A *Turing Machine* T is a function

$$T : \mathbb{N} \to \mathbb{N} \cup \{\bot\}$$

that is agreed to be computable.

Now, note that we can define a one-to-one mapping between the set of all finite binary strings $\mathbb{B}^* = \{0,1\}^*$ and the natural numbers by the relation induced by a lexicographic order of the form:

$$\{(\text{``''}, 0), (\text{``0''}, 1), (\text{``1''}, 2), (\text{``00''}, 3), (\text{``01''}, 4)\dots\}.$$

Using this relation we can see all natural numbers (or positive integers) as binary strings and vice versa. Accordingly, every finite binary string has a natural number associated to it. Therefore the definition 2.1 is equivalent to:

Definition 2.2. A *Turing Machine* T is a function

$$T : \mathbb{B}^* \to \mathbb{B}^* \cup \{\perp\}$$

that is agreed to be computable.

A string $p \in \mathbb{B}^*$ is defined as a *valid program* for the Turing machine T if, during the execution of T with p as input, all the characters in p are read. The resulting binary string of evaluating $T(p)$ is called the *output* of the machine. If the output is $\perp$ then we say the machine *failed to stop*. A Turing Machine is *prefix-free* if no valid program can be a proper substring of another valid program (though it can be a postfix of one)[1]. We call a valid program a *self-delimited object*. Note that, given the relationship between natural numbers and binary strings, the set of all valid programs is an infinite proper subset of the natural numbers.

2.1.1 Computable Functions and Objects

In a broad sense, an object x is *computable* if it can be *described* by a Turing Machine: if there exists a Turing machine that produces x as an output given 0 as an input (or the empty string). It is trivial to show that any finite string on a finite alphabet is a computable object. Now we have all the elements to formally define more complex computable objects.

Definition 2.3. A function $f : \mathbb{N} \to \mathbb{N}$ is *computable* if there exists a Turing Machine T such that $f(x) = T(x)$.

The following statements can be easily proven for Turing machines and λ-calculus, but I will enunciate them as **axioms**. Let f and g be computable functions, then

- the composition $f(g(x))$ is a computable,
- if $g(x) = \perp$ then $f(g(x)) = \perp$ and

[1]A string is a prefix if it can be found at the begging of a bigger string and is postfix if it can be found at the end.

- the string concatenation $f(x)g(x)$ (likewise denoted $f(x) \cdot g(x)$) is also computable.

Using computable functions as a base, we can easily describe more mathematical objects.

For **functions with more than one variable**, if x is a pair $x = (x_1, x_2)$, we say that the codification $g(x)$ is unambiguous if it is injective and the inverse functions $g_1^{-1} : g(x) \mapsto x_1$ and $g_2^{-1} : g(x) \mapsto x_2$ are computable. If x is a tuple $(x_1, ..., x_i, ..., x_n)$, then the codification $g(x)$ is unambiguous if the function $(x, i) \mapsto x_i$ is computable. Having defined computable functions over several variables, is easy to see that, over computable objects, all the Peano arithmetic operators are also computable.

A **sequence** of natural numbers (or binary strings) $\delta_1, \delta_2, ..., \delta_i, ...$ is computable if the function $\delta : i \mapsto \delta_i$ is computable. A **real number** is computable if its decimal expansion is a computable sequence. For complex numbers and higher dimensional spaces, we say that they are computable if each of their coordinates are also computable. Finally, an **infinite set** S over the natural numbers is computable if there exists a computable function d such that $x \in S$ if and only if $d(x) = 1$.

For further extending the notion of computability to more complex mathematical objects is very useful to introduce the notion of universal machine:

Definition 2.4. A Turing Machine U is considered *universal* if there exists a computable function g such that for every Turing machine T there exists a string $\langle T \rangle \in \mathbb{B}^*$ such that $T(x) = U(\langle T \rangle g(x))$, where $\langle T \rangle g(x)$ is the concatenation of the strings $\langle T \rangle$ and $g(x)$. Given the previous case, $\langle T \rangle$ and $g(x)$ are called a *codification or a representation* of the function T and the natural number x, respectively. From now on I will denote the codification of T and x by $\langle T \rangle$ and $\langle x \rangle$ respectively. The codification $g(x)$ is *unambiguous* if it is injective.

When there exists $x \in \mathbb{N}$ such that $T(x) = \perp$, it is understood that T *does not stop or diverges in* x. Turing seminal results shows that there exists universal Turing machines, but there does not exists a Turing machine T_U such that, for a universal machine U and a given machine T, $T_U(\langle T \rangle \langle x \rangle) = 1$

if and only if $T(x) \neq \bot$. This is called the *undecidability* of the halting set HP, also known as the Halting problem (HP).

Definition 2.5. In general, a set S is *undecidable* if it does not exists a computable function d such that $x \in S$ if and only if $d(x) = 1$. A set is *semi-decidable* or *semi-computable* if there exists a computable function d such that $d(x) = 1$ implies $x \in S$. A set is decidable if it is computable. The term 'decidable' is used since the function d *decides* a set membership problem.

Now, for each of the objects described, we refer to the representation of the associated Turing machine as *the representation of the object for the reference Turing machine U*, and we define the computability of further objects by considering their representations. For example, a function $f : \mathbb{R} \to \mathbb{R}$ is computable if the mapping $\langle x_i \rangle \mapsto \langle f(x_i) \rangle$ is computable and we will denote by $\langle f \rangle$ the representation of the associated Turing machine, calling it the **codification** of f itself.

The definition of computability for further mathematical objects, such as infinite sets of real numbers and sets of sets, follows from working with representations. Working this way, it is easy to see how we can build a framework for all computable mathematics and, therefore, computable science in general.

It is the personal opinion of the author that all science that is currently used to calculate, model or predict the natural world is computable. And even when working with incomputable objects, we rely on computable *bridges* to reach objective conclusions, such as the rules of first oder logic. I reached this conclusion given that formal systems based on logical principles are the fundation of modern mathematics, and logical derivation is a computable process (for most widely used formalisms).

2.2 Algorithmic Information

Defined independently by Chaitin [18], Solomonoff [70] and Kolmogorov [47], the notion of the algorithmic information content of a (computable) object measures the minimum amount of information that is needed to fully describe the object without ambiguities. This description is done within the framework of Turing machines since they define everything that can be described without ambiguity given a finite set of rules. Formally,

Definition 2.6. Given a prefix-free[2] universal Turing Machine U with alphabet Σ, the *algorithmic descriptive complexity* (also known as Kolmogorov complexity and Kolmogorov-Chaitin complexity [47, 20]) of a string $s \in \Sigma^*$ is defined as

$$K_U(s) = \min\{|p| : U(p) = s\},$$

where U is a universal prefix-free Turing Machine and $|p|$ is the number of characters of p.

The stated function measures the minimum amount of information needed to fully describe a computable object within the framework of a universal Turing machine U. If $U(p) = s$ then the program p is called a *description* of s. The first of the smallest descriptions (in alphabetical order) is denoted by s^*, and by $\langle s \rangle$ a not necessarily minimal description computable over the class of objects. If M is a Turing machine, a program p is a description or codification of M for U if for every string s we have it that $M(s) = U(p \cdot g(s))$, where g is an unambiguous computable function.

In the case of **numbers**, **functions**, **sequences**, **sets** and other computable objects we consider the descriptive complexity of their smallest descriptions. For example, given a computable function $f : \mathbb{R} \to \mathbb{R}$, $K_U(f)$ is defined as $K_U(f^*)$, where $f^* \in \mathbb{B}^*$ is the first of the minimal descriptions for f.

Definition 2.7. Of particular importance for this document is the *conditional descriptive complexity of s with respect to r*, which is defined as:

$$K_U(s|r) = \min\{|p| : U(pr) = s\},$$

where pr is the concatenation of p and r.

Conditional descriptive complexity can be interpreted as the *smallest amount of information needed to describe s given a full description of r.* We can think of p as a program with input r.

[2]See definition 2.2 and next subsection.

One of the most important properties of the descriptive complexity measure is its *stability or invariability*: the difference between the descriptive complexity of an object, given two universal Turing machines, is at most constant.

Theorem 2.8. *Invariance theorem [18, 47, 70]: Let U_1 and U_2 be two Universal Turing machines and $K_{U_1}(s)$ and $K_{U_1}(s)$ the algorithmic descriptive complexity of s for U_1 and U_2, respectively; there exists a constant $c_{U_1U_2}$ such that*

$$|K_{U_1}(s) - K_{U_1}(s)| \leq c_{U_1U_2},$$

where $c_{U_1U_2}$ is a constant that depends on U_1 and U_2 but not on s. This constant can be understood as the length of a program, called compiler*, that translates any program p for U_1 to a program p' for U_2 and vice versa.*

It is this theorem that gives to K the adjective of *universal*: the algorithmic complexity of an object is independent of its reference, except for a small error. Referencing the underlying universal machine U is usually omitted in favor of the *universal measure* K. From now on I will too omit the subscript.

On Self-delimited Objects

Now is a good point to note that I am defining K for **prefix-free** universal machines. This property is important since it allows us to *daisy chain*[3] programs with **minimal impact** on the required program length. First, let me explain the nature of self-delimited objects.

Recall that a **self delimited object** was defined as a valid program for our reference universal machine. Since I am only considering prefix-free programs, then, for any finite sequence of valid (prefix-free) programs $p_1, p_2, \ldots, p_n$, we can describe a program q such that, using the description of an universal machine, can execute the first program p_1 until it reads all of its characters, repeating the process n times. It follows that if the sequence $p_1, p_2, \ldots, p_n$ describes n different objects, then the algorithmic complexity

[3]To concatenate programs such that the input of the second is the output of the first.

of the set of all the objects is equal (or less) than the sum

$$|q| + \sum_{i}^{n} |p_i|.$$

Describing the same set for a reference universal machine with prefixes — using a similar program— we would incur into a non constant penalty, as shown in the case of the next paragraph.

Now, lets us consider what is needed to describe any natural number (or string) x. One might use the *identity* Turing machine $I(x) = x$. However, simply concatenating the strings, $\langle I \rangle x$, doest not yield a valid codification for the program and its input, otherwise the string $\langle I \rangle x0$ would also be a valid program and our reference universal Turing machine would not be prefix-free. To solve this issue we must find a self contained form for x:

Definition 2.9. The natural self-delimited encoding of a finite string s is defined as the string

$$1^{\log(|s|)}0s.$$

This is a string of 1's of length $\log(|s|)$, followed by a 0 and the string itself [53, section 1.4]. For $\log(n)$ we will always understand the rounding up of the logarithm base 2 of n.

Given the previous encoding of s, we can now describe a program that reads the 1's until encountering the first 0. With this the program now has the length of the string s and can read it without going outbounds. It follows that for any binary string s (or natural number s):

$$K(s) \leq \log(|s|) + |s| + O(1) = O(\log(s)).$$

2.2.1 Randomness

We now have all the elements to solve the problem introduced at the beginning of the chapter and give a definition of randomness: a string is random when it cannot be described by a short program [70].

Definition 2.10. Given a natural number r, a string s is considered *r-random* or *incompressible* if $K(s) \geq |s| - r$. This definition would have it that a string is random if it does not have a significantly shorter complete description than the string itself.

A simple counting argument shows the existence of random strings: For all n, there are 2^n strings of length n; however the number of shorter descriptions is

$$2^n - 1 = \sum_{i=0}^{n-1} 2^i,$$

therefore there must me be at least one random string for any length. Furthermore, recall that not all strings are valid programs and that there are many descriptions for the same program, therefore the number of random strings is even higher. In fact, the ratio of random to compressible strings goes to 1 as n goes to infinity for any r. In other words, there are many more random strings than non-random ones.

Definition 2.11. Recall the self-delimited encoding of s. It follows that there exists a natural r such that if s is r-random then

$$K(x) = |x| - r + O(\log |x|), \tag{2.1}$$

where $O(\log |x|)$ is a positive term. We will say that for such strings the randomness inequality holds *tightly*.

Let M be a halting Turing Machine with description $\langle M \rangle$ for the reference machine U. A simple argument can show that the halting time of M cannot be a large *random* number: Let U^H be a Turing Machine that emulates U while counting the number of steps, returning the execution time upon halting. If r is a large random number, then M cannot stop in time r, otherwise the program $\langle U^H \rangle \langle M \rangle$ will give us a *short* description of r. This argument is summarized by the following inequality:

$$K(T(M)) \leq K(M) + O(1), \tag{2.2}$$

where $T(M)$ is the number of steps that it took the machine M to reach the halting state, the *execution time* of the machine M.

It is important to note that the inequality 2.2, along with the construction given in definition 2.9, directly imply that K **is a no computable (incomputable) function**, otherwise the self-delimiting encoding for natural numbers would set a hard upper bound to the execution time for any halting machine, solving HP.

As a consequence, we can say that incomputable objects, like the set HP, are *irreducible*, given that there is no *short* (finite) computable descriptions for them. I characterrize this kind of objects as containing an *infinite ammount of information*.

2.3 Complexity

Diverse scientific fields have their own notions and measures of complexity. For example, in statistics, the complexity of a linear regression is often defined as the degree of the polynomial model; for topology the Hausdorff dimension and in dynamical systems theory we have the notion of chaos. In the opinion of this author, the most accomplished definition of complexity is found in AIT, as precludes the others by virtue of being established at the core of what can be computed. It is natural to expect that the degree of a polynomial or the dimension of a space to be highly correlated to the size of the minimum description of the respective object, or with respect to the complexity of the input in the case of chaotic systems (systems as in [37]). Furthermore, K is capable of detecting regularities that the previous measure of complexity are likely to omit.

However, K by itself is not a good measure of what is intuitively deemed complex: in scientific literature random objects are considered to be as simple as highly regulars ones [8, 48, 35, 1, 76]. In the words of physicist Murray Gell-Marin [35]:

> "A measure that corresponds much better to what is usually meant by complexity in ordinary conversation, as well as in scientific discourse, refers not to the length of the most concise description of an entity (which is roughly what AIC is), but to the length

of a concise description of a set of the entity's regularities. Thus something almost entirely random, with practically no regularities, would have effective complexity near zero."

From a computability point of view, the argument states that random strings are as simple as very regular strings, given that there is no complex underlying structure in their minimal descriptions. The intuition that random objects contain no useful information leads us to the same conclusion. Thus, we need measures of complexity that can classify random objects (or their respective computable representations) as simple, along with highly regular objects.

It turns out that measures with this property can be also be found in AIC, using algorithmic information content at is core, and thus they are also *universal*.

2.3.1 Universal Measures of Complexity

Sophistication is a measure of *useful information* within a string. Proposed by Koppel [48], the underlying approach consists in dividing the description of a string x into two parts: the program that represents the *underlying structure* of the object, and the input, which is the random or *structureless* component of the object. This function is denoted by $soph_c(x)$, where c is a natural number representing the significance level.

Definition 2.12. The *sophistication* of a natural number x at the significance level c, $c \in \mathbb{N}$, is defined as:

$$\begin{aligned} soph_c(x) = \min\{|\langle p\rangle| : \text{p is a total function and} \\ \exists y.p(y) = x \text{ and } |\langle p\rangle| + |y| \leq K(x) + c\} \end{aligned}$$

Now, the images of a mapping $\delta : i \mapsto \delta_i$ already have the form $\delta(i)$, where δ and i represent the structure and the *random* component respectively. Random inputs i should bind this structure strongly up to a logarithmic error, which is proven in the next lemma.

Lemma 2.13. *Let $\delta_1, ..., \delta_i, ...$ be a sequence of different natural numbers and r a natural number. If the function $\delta : i \mapsto \delta_i$ is computable then there exists an infinite subsequence where the sophistication is bounded up the logarithm of a logarithmic term of their indices.*

Proof. Let δ be a computable function. Note that since δ is computable and the sequence is composed of different naturals, its inverse function δ^{-1} can be computed by a program m which, given a description of δ and δ_i, finds the first i that produces δ_i and returns it; therefore

$$K(i) \leq K(\delta_i) + |\langle m \rangle| + |\langle \delta \rangle|$$

and

$$K(\delta) + K(i) \leq K(\delta_i) + |\langle m \rangle| + 2|\langle \delta \rangle|.$$

Now, if i is a r-random natural where the inequality holds tightly, we have it that

$$(K(\delta) + O(\log |i|)) + |i| - r \leq K(\delta_i) + |\langle m \rangle| + 2|\langle \delta \rangle|,$$

which implies that, since δ is a total function,

$$soph_{(|\langle m \rangle| + 2|\langle \delta \rangle| + r)}(\delta_i) \leq K(\delta) + O(\log \log i).$$

Therefore, the sophistication is bounded up to the logarithm of a logarithmic term for a constant significance level for an infinite subsequence. $\square$

Small changes in the significance level of sophistication can have a large impact on the sophistication of a given string. Another possible issue is that the constant proposed in lemma 2.13 could appear to be large at first (but it becomes comparatively smaller as i grows). A *robust* variation of sophistication called coarse sophistication [4] incorporates the significance level as a penalty. The definition presented here differs slightly from theirs in order to maintain congruence with the chosen prefix-free universal machine and to avoid negative values. This measure is denoted by $csoph(x)$.

Definition 2.14. The *coarse sophistication* of a natural number x is defined as:

$$csoph(x) = \min\{2|\langle p \rangle| + |\langle y \rangle| - K(x) : p(y) = x \text{ and } p \text{ is total}\},$$

where $|\langle y \rangle|$ is a computable unambiguous codification of y.

With a similar argument as the one used to prove lemma 2.13, it is easy to show that coarse sophistication is similarly bounded up to the logarithm of a logarithmic term.

Lemma 2.15. *Let $\delta_1, ..., \delta_i, ...$ be a sequence of different natural numbers and r a natural number. If the function $\delta : i \mapsto \delta_i$ is computable, then there exists an infinite subsequence where the coarse sophistication is bounded up to the logarithm of a logarithmic term.*

Proof. If δ is computable and i is r-random, then by definition of *csoph* and the inequalities presented in the proof of lemma 2.13, we have that

$$\begin{aligned} csoph(\delta_i) \leq & 2K(\delta) + (|i| + 2\log|i| + 1) - K(\delta_i) \\ \leq & 2K(\delta) + (|i| + 2\log|i| + 1) - K(i) + |\langle M\rangle| + |\langle\delta\rangle| \\ \leq & 2K(\delta) + |\langle M\rangle| + |\langle\delta\rangle| + (|i| + 2\log|i| + 1) - |i| + r \\ = & 2K(\delta) + |\langle M\rangle| + |\langle\delta\rangle| + r + 1 + O(\log\log i)) \end{aligned}$$

□

Another proposed measure of complexity is Bennett's logical depth [8], which measures the minimum computational time required to compute an object from a nearly minimal description. Logical depth works under the assumption that complex or *deep* natural numbers take a long time to compute from near minimal descriptions. Conversely, random or incompressible strings are shallow since their minimal descriptions must contain the full description *verbatim*.

Definition 2.16. The *logical depth* of a natural x at the level of significance c is defined as:

$$depth_c(x) = \min\{T(p) : |p| - K(x) \leq c \text{ and } U(p) = x\}.$$

where $T(P)$ is the halting time of the program p.

Working with computing times has shown to be notoriously difficult and often related to open fundamental problems in computer science and mathematics. For instance, finding a *low* upper bound to the growth of logical depth of all computable series of natural numbers would suggest a negative answer to the question of the existence of an efficient way of generating deep strings, which Bennett relates to the open $P \neq PSPACE$ problem. For this reason, I will use a related measure called busy beaver logical depth, denoted by $depth_{bb}(x)$.

Definition 2.17. The *busy beaver logical depth* of the description of a natural x, denoted by $depth_{bb}(x)$, is defined as:

$$depth_{bb}(x) = \min\{|p| - K(x) + j : U(p) = x \text{ and } T(p) \leq BB(j)\},$$

where $BB(j)$, known as the busy beaver function, is the halting time of the slowest program that can be described within j bits [28].

Instead of working directly with computing time, this version of logical depth adds a penalty value according busy beaver function: if $T(p)$ is large, then the smallest j such that $BB(j) \geq T(p)$ is also expected to be large. With this, the measure also manages to avoid the significance value of the original logical depth function.

Definition 2.18. Formally, the busy beaver function [62] for a natural n is defined as:

$$BB(n) = \max\{T(U(p)) : |p| \leq n\}.$$

This function is important for AIT and computer science in general as it defines a limit to the execution time **and** the descriptive power of an algorithm in terms of its description in form of a growing, uniform function. As with K itself and the other complexity measures mentioned in this chapter, $BB(n)$ is incomputable.

The next result follows from a theorem formulated by Antunes and Fortnow [4] and from lemma 2.15.

Corollary 2.19. *Let $\delta_1, ..., \delta_i, ...$ be a sequence of different natural numbers and r a natural number. If the function $\delta : i \mapsto \delta_i$ is computable, then there exists an infinite subsequence where the busy beaver logical depth is bounded up to the logarithm of a logarithmic term of their indeces.*

Proof. By theorem 5.2 at [4], for any i we have that

$$|csoph(\delta_i) - depth_{bb}(\delta_i)| \leq O(\log |\delta_i|).$$

By lemma 2.15 and theorem 4.1 the result follows. □

On the Heuristics Behind AIT Complexity Measures

By separating the structured and unstructured data and defining the complexity in terms of the algorithmic description of the first, sophistication (and coarse sophistication) manages to robustly capture the heuristics behind commonly used complexity measures in science for computable objects.

For example, the theory behind a linear regression models describes a —presumed infinite— set S through a polynomial function f. This function is the structured part of the set. Ideally, the points are meant to be normally distributed with center at f; the parameter for this normal distribution is the irreducible information. It follows that a computable description of the set is $\langle f\rangle\langle\sigma\rangle$ and, if the linear model is indeed the best way to describe S, then the sophistication of the set is given by $K(f)$, within a constant error. If there exists a better model, then $soph_c(S) < K(f)$. Furthermore, by defining the complexity of the set in terms of the minimal descriptions of f, we are also measuring the complexity of the coefficients themselves, ultimately offering a more robust complexity measure than just the degree of the resultant polynomial. This reasoning can be extended to other commonly used complexity measures.

Chapter 3

Computable Dynamical Systems: Evolution

Broadly speaking, a dynamical system is one that changes over time. Prediction of the future behavior of dynamical systems is a fundamental concern of science in general. Scientific theories are tested upon the accuracy of their predictions, and establishing invariable properties through the evolution of a system is an important goal.

Limits to this predictability are known in science. For instance, chaos theory establishes the existence of systems in which small deficits in the information of the initial states makes accurate predictions of future states unattainable. However, I will focus on systems for which we have unambiguous, finite (as to size and time) and complete descriptions of initial states and behavior: computable dynamical systems.

Since their formalization by Church and Turing, the class of computable systems has shown that, even without information deficits (i.e., with complete descriptions), there are future states that cannot be predicted, in particular the state known as the *halting state* [74]. In this section I will use this result and others from AIT (algorithmic information theory) to show how predictability imposes limits to the growth of complexity during the evolution of computable systems. In particular, I will show that random (incompressible) times tightly bound the complexity of the associated states, ultimately imposing limits to the open-endedness of decidable evolutionary systems.

3.1 Computable Evolutionary Systems

The simplest, most unassuming definition that the author can give for what I understand as *(Darwinian) evolution* is the observed production of new kind of organisms over time, from a presumed initial state, driven by *adaptation*. If we assume that this process can be modeled by the language of science, then we should be able to describe it as a **dynamical system**. Based on this minimal description, I will formulate a rigorous mathematical definition using AIT that captures any computable and deterministic theory of evolution.

A *dynamical system* is a rule of evolution in time within a state space; a space that is defined as the set of all possible states of the system [57]. I will focus on a functional model for dynamical systems with a constant initial state and variables representing the previous state and the time of the system. This model allows me to set halting states for each time on a discrete scale in order to study the impact of the descriptive complexity of time during the evolution of a discrete computable system. Therefore, within the language of a functional dynamical system:

Definition 3.1. A *deterministic evolutionary system with discrete time* is a function $S(M_0, t, E(t)) : \mathbb{N} \mapsto \mathbb{N}$ where there is:

- An *evolution function (or rule)* of the form $M_{t+1} = S(M_0, t, E(t))$.
- A *time* variable $t \in \mathbb{N}$.
- An *initial state* M_0 such that $S(M_0, 0, E(0)) = M_0$.
- An *environment* E that changes over time and affects the evolution of S.

If $t \mapsto M_t$ is a computable function and M_0 is a computable object, we will say that S is a *computable dynamical system.*

The sequence of states $M_0, M_1, ..., M_t, ...$ is called the evolution of the system, where M_t is the state of the system at the time t. Given that M_t and $S(M_0, t, E(t))$ define the same object, I will use them indistinctly, depending on which representation offers more clarity to the reader. Also, is possible to define a function S' such that $S'(M_0, t) = S(M_0, t, E(t))$, therefore I will use the shorthand $S(M_0, t)$ when the environment function $E(t)$ is not important or when the discussion can be extended to dynamical systems in general.

One important property for the defined class of systems is the **uniqueness of the successor state** property: given that the system is assumed to be deterministic, we have that equal states must evolve equally given the same evolution function. In other words:

$$M_t = M_{t'} \implies M_{t'+1} = M_{t+1}. \tag{3.1}$$

The converse is not necessarily true. It is also important to note that we are implying an infinity of possible states for non-cyclical systems.

Now, a *complete description of a computable dynamical system* $S(M_0, t)$ should contain enough information to compute the state of the system at any time; hence it must entail the codification of its evolution function S and a description of the initial state M_0, which is denoted by $\langle M_0 \rangle$. As a consequence, if we only describe the system at time t by a codification of M_t, then we would not have enough information to compute the successive states of the system. So I will specify the *complete description* of a computable system at time t as an unambiguous codification of the ordered pair composed of $\langle S \rangle$ and $\langle M_t \rangle$, i.e. $\langle (S, \langle M_t \rangle) \rangle$, with $\langle (S, \langle M_0 \rangle) \rangle$ representing the initial state of the system. It is important to note that, for any computable and unambiguous codification function g of the stated pair, we have

$$K(\langle (S, \langle M_t \rangle) \rangle) \leq K(S) + K(M_0) + K(t) + O(1),$$

as we can write a program that uses the descriptions for S, M_0 and t to find the parameters and then evaluate $S(M_0, t)$, finally producing M_t.

3.2 Open-Ended Evolution: A Formal Definition

As mentioned in the introduction, **open-ended evolution (OEE)** has been characterized as "evolutionary dynamics in which new, surprising, and sometimes more complex organisms and interactions continue to appear" [73] and is considered a fundamental property for systems capable of producing life [63]. This has been implicitly verified by various experiments *in-silico* [54, 2, 51, 5].

One line of thought posits that open-ended evolutionary systems tend to produce families of objects of increasing *complexity* [6, 5]. Furthermore, for a number of complexity measures, it can be shown that the objects belonging to a given level of complexity are finite (for instance $K(x)$ and logical depth) and for others, the dynamics (or structured information) are finite (such as sophistication). Therefore an increase of complexity is a requirement for the continued production of new objects with *innovative interactions*. A related observation, proposed by Chaitin [21, 23], associates evolution with the search for *mathematical creativity*, which implies an increase of complexity, as more complex mathematical operations are needed in order to solve *interesting problems*, which are *required to drive evolution*.

Following the aforementioned lines of thought, I have chosen to characterize OEE in computable dynamical systems as a process that has the property of producing families of objects of increasing complexity. Formally, given a *complexity measure* C, we say that a computable dynamical system S exhibits *open-ended evolution* with respect to C if for every time t there exists a time t' such that the complexity of the system at time t' is greater than the complexity at time t, i.e. $C(S(M_0,t)) < C(S(M_0,t'))$, where a complexity measure is a (not necessarily computable) function that goes from the state space to a positive numeric space.

The existence of such systems is trivial for complexity measures on which any infinite set of natural numbers (not necessarily computable) contains a subset where the measure grows strictly:

Lemma 3.2. *Let C be a complexity measure such that any infinite set of* *natural numbers has a subset where C grows strictly. Then a computable system $S(M_0, t)$ is a system that produces an infinite number of different states if and only if it exhibits OEE for C.*

Proof. Let $S(M_0, t)$ be a system that does not exhibit OEE, and C a complexity measure as described. Then there exists a time t such that for any other time t' we have $C(M'_t) \leq C(M_t)$, which holds true for any subset of states of the system. It follows that the set of states must be finite. Conversely, if the system exhibits OEE, then there exists an infinite subset of states on which S grows strictly, hence an infinity of different states. $\square$

Given the previous lemma, a trivial computable system that simply produces all the strings in order exhibits OEE on a class of complexity measures that includes algorithmic description complexity, sophistication and logical depth. However, we intuitively conjecture that such systems have a much simpler behavior compared to that observed in the natural world and non-trivial artificial life systems. To avoid some of these issues I propose a stronger version of OEE.

Definition 3.3. A sequence of natural numbers $n_0, n_1, \ldots, n_i, ...$ exhibits *strong open-ended evolution* (strong OEE) with respect to a complexity measure C if for every index i there exists an index i' such that $C(n_i) < C(n_{i'})$, and the sequence of complexities $C(n_0), C(n_1), \ldots, C(n_i), \ldots$ does not drop *significantly*, i.e. there exists γ such that $i \leq j$ implies $C(n_i) \leq C(n_j) + \gamma(j)$ where $\gamma(j)$ is a positive function such that $C(n_j) - \gamma(j)$ is not upper-bounded for any infinite subsequence $\gamma(n_{i_j})$ where the strong OEE inequality holds.

It is important to note that while the definition of OEE allows for significant drops in complexity during the evolution of a system, strong OEE requires that the complexity of the system does not decrease *significantly* during its evolution. In particular we will require that the *complexity drops*, as measured by γ, not *grow as fast as the complexity itself* and that they do not reach a constant level an infinite number of times.

Finally, I will construct the concept of *speed* of growth of complexity in a comparative way: given two sequences of natural numbers n_i and m_i, n_i *grows faster* than m_i if for every infinite subsequence and natural number N, there exists j such that $n_i - m_j \geq N$. Conversely, a subsequence of indexes denoted by i grows faster than a subsequence of indexes denoted by j if for every natural N, there exists i with $i < j$, such that $n_i - n_j \geq N$.

If a complexity measure is sophisticated enough to depend on more than just the size of an object, significant drops in complexity are a feature that can be observed in trivial sequences such as the ones produced by enumeration machines. Whether this is also true for *non-trivial* sequences is open to debate. However, by classifying random strings as low complexity objects and posit that non-trivial sequences must contain a limited number of random objects, then a non-trivial sequence must observe bounded drops in complexity in order to be capable of showing non-trivial OEE. This is the intuition behind the definition of strong OEE.

In resume, my definition of strong-OEE characterizes open-ended evolution if we assume:

- A space of computable, deterministic dynamical systems with discrete time.
- Random (incompressible) objects are simple.
- OEE implies unbounded a growth in complexity.
  - This growth does not need to be monotonous.
- The growth in complexity meets a baseline of **stability**.

3.3 A Formal Characterization for Adaptation

It has been argued that in order for *adaptation* and survival to be possible an organism must contain an effective representation of the environment, so that, given a reading of the environment, the organism can choose a behavior accordingly [79]. The more approximate this representation, the better the adaptation. If the organism is computable, this information can be codified by a computable structure. We can consider M_t, where t stands for the time corresponding to each of the stages of the evolution, a description of an organism or population. This information is then processed following a finitely specified unambiguous set of rules that, in finite time, will determine the adapted behavior of the organism according to the information codified by M_t; the adapted behavior can represent a simulation or a (computable) theory explaining an organism. I will denote this behavior using the program p_t. An adapted system is one that produces an acceptable approximation of its environment. An environment can also be represented by a computable structure or function E. In other words, the system is adapted if $p_t(M_t)$ produces $E(t)$.

Based on this idea I propose a robust, formal characterization for adaptation:

Definition 3.4. Let K be the prefix-free descriptive complexity. We say that the system at the state M_n is *ϵ-adapted* to the E if:

$$K(E|S(M_0, t, E(t))) \leq \epsilon. \tag{3.2}$$

The inequality states that the minimal amount of information that is needed to describe E from a complete description of M_t is ϵ or less. This information is provided in the form of a program p that produces $E(t)$ from the system at time t. We will define such a program p as the *adapted behavior* of the system. It is not required that p be unique.

The proposed structure to characterize adapted states is robust since $K(E|S(M_0, t, E(t)))$ is less than or equal to the number of characters needed to describe any computable method of describing E from the state of the system at time t, whether it be a computable theory for adaptation or a computable model for an organism that tries to predict E. It follows that any computable characterization of adaptation that can be described within ϵ number of bits meets the definition of *ϵ-adapted*, given a suitable choice of E, the *adaptation condition* for any given environment.

It is important to note that, although inspired by a representationalist approach to adaptation, the proposed characterization of adaptation is not contingent on the organisms containing an actual codification of the environment, since any organism that can produce an adapted behavior that can be explained effectively (that is, it is computable in finite time) is ϵ-adapted for some ϵ. This also implies that ***ϵ-adapted* is a necessary, but not sufficient condition of adaptation**, hence I call it characterization for adaptation, rather than a definition.

As a simple example, we can think of an organism that must find food located at the coordinates (x, j) on a grid in order to survive. If the information in an organism is codified by a computable structure M (such as DNA), and there is a set of finitely specified, unambiguous rules that govern how this information is used (such as the ones specified by biochemistry and biological theories), codified by a program p, then we say that the organism finds the food if $p(M) = (j, k)$. If $|\langle p \rangle| \leq \epsilon$, then we say that the organism is adapted according to a behavior that can be described within ϵ characters.

The proposed model for adaptation is not limited to such simple interactions. For a start, we can suppose that the organism *sees* a grid, denoted by g, of size $n \times m$ with food at the coordinates (j, k). The environment can be codified as a function E such that $E(g) = (j, k)$ and ϵ-adapted implies that the organism defined by the genetic code M, which is interpreted by a theory or behavior written on ϵ bits, is capable of finding the food upon *seeing* g. Similarly, more complex computational structures and interactions imply ϵ-adaptation.

Now, for fixed environments, describing an evolutionary system that (eventually) produces an *ϵ-adapted* system is trivial via an enumeration machine (the program that produces all the natural numbers in order), as it will eventually produce E itself. Moreover, we require the output of our process to remain adapted. Therefore I propose a stronger condition called *convergence*:

Definition 3.5. Given the description of a computable dynamical system $S(M_0, t, E)$ where $t \in \mathbb{N}$ is the variable of time, M_0 is an initial state and E is an environment, we say that the system S *converges* towards E with degree ϵ if there exists δ such that $t \geq \delta$ implies $K(E|S(M_0, t, E)) \leq \epsilon$.

For a fixed initial state M_0 and environment E, it is easy to see that the descriptive complexity of a state of the system depends mostly on t: we can describe a program that, given full descriptions of S, E, M_0 and t, finds

$S(M_0, t, E(t))$. Therefore

$$K(S(M_0, t, E(t))) \leq K(S) + K(E) + K(M_0) + K(t) + O(1), \tag{3.3}$$

where the constant term is the length of the program described. In other words, as the time t grows, *time becomes the main driver for the descriptive complexity within the system.*

Chapter 4

Irreducibility of Descriptive Time Complexity

In the previous chapter, I showed that time was the main factor in the descriptive complexity of the states within the evolution of a system. This result is expanded by the *time complexity stability theorem* (4.1). This theorem establishes that, within an algorithmic descriptive complexity framework, similarly complex initial states must evolve into similarly complex future states over similarly complex time frames, effectively erasing the difference between the complexity of the state of the system and the complexity of the corresponding time, establishing absolute limits to the reducibility of future states.

Let $F(t) = T(S(M_0, t, E(t)))$ be the *real execution time* of the system at time t. Using our time counting machine U^H, it is easy to see that $F(t)$ is computable and, given the uniqueness of the successor state, F increases strictly with t, and hence is injective. Consequently, F has a computational inverse F^{-1} over its image. Therefore, we have it that (up to a small constant) $K(F(t)) \leq K(F) + K(t)$ and $K(t) \leq K(F^{-1}) + K(F(t))$. It follows that

$$K(t) = K(F(t)) + c,$$

where c is a constant independent of t (but that can depend on S). In other words, for a fixed system S, the execution time and the system time are *equally complex up to a constant.* From here on I will not differentiate between the complexity of both times. A generalization of the previous equation is given by the following theorem:

Theorem 4.1 (Time Complexity Stability)**.** *Let S and S' be two computable systems and t and t' the first time where each system reaches the states M_t and $M'_{t'}$ respectively. Then there exists c such that $|K(M_t) - K(t)| \leq c$ (analogous for t') and $|K(M_t) - K(M'_{t'})| \leq c$. Specifically:*

i) There exists a natural number c that depends on S and M_0, but not on t, such that

$$|K(M_t) - K(t)| \leq c. \tag{4.1}$$

ii) If

$$K(S(M_0, t, E(t))) = K(S'(M'_0, t', E'(t))) + O(1)$$

and

$$K(M_0) = K(M'_0) + O(1)$$

then there exists a constant c that does not depend on t such that

$$|K(t) - K(t')| \leq c,$$

where t and t' are the minimum times for which the corresponding state is reached.

iii) Let S and S' be two dynamical systems with an infinite number of equally–up to a constant–descriptive complex times α_i and δ_i. For any infinite subsequence of times with strictly growing descriptive complexity, all but finitely many j, k such that $j > k$ comply with the equation:

$$K(\alpha_k) - K(\alpha_j) = K(\delta_k) - K(\delta_j).$$

Proof. First, note that we can describe a program such that given S, M_0 and E, runs $S(M_0, E, x)$ for each x until it finds t. Therefore

$$K(t) \leq K(S(M_0, E, t)) + K(S) + K(M_0) + K(E) + O(1). \tag{4.2}$$

Similarly for t'. By the inequality 3.3 and the hypothesized equalities we obtain

$$\begin{aligned} K(t) - (K(S) + K(M_0) + K(E) + O(1)) & \\ \leq K(M_t) & \\ \leq K(t) + (K(S) + K(E) + K(M_0) + O(1)), & \end{aligned}$$

which implies the first part. The second part is a direct consequence.

For the third part, suppose that there exists an infinity of times such that

$$K(\alpha_k) - K(\alpha_j) > K(\delta_k) - K(\delta_j).$$

Therefore

$$K(\alpha_k) - K(\delta_k) > K(\alpha_j) - K(\delta_j),$$

which implies that the difference is unbounded, which is a contradiction of the first part. Analogously, the other inequality yields the same contradiction. □

The **slow growth** of time is a possible objection to the assertion that in the descriptive complexity of systems time is the dominating parameter for predicting their evolution: the function $K(t)$ grows within an order of $O(\log t)$, which is very slow and often considered insignificant in the information theory literature. However, we have to consider the scale of time we are using. For instance, one second of *real time* in the system we are modeling may mean an exponential number of discrete time steps for our computable model (for instance, if we are emulating a genetic machine with current computer technology), yielding a potential polynomial growth in their descriptive complexity. Nevertheless, if this time conversion is computable, then $K(t)$ grows at most at a constant pace. This is an instance of *irreducibility*, as there exist an infinity sequences of times that cannot be obtained by computable methods. This time instances will be known as *random times* and the sequences containing them will be deemed **irreducible**.

4.1 Non-Randomness of Decidable Convergence Times

As mentioned before, one of the most important issues for science is predicting the future behavior of dynamical systems. The prediction I will focus on is about the first state of convergence (definition 3.5): Will a system converge and how long will it take? In this section I shall show the limit that decidability imposes on the complexity of the first convergent state. A consequence of this is the existence of undecidable adapted states.

Formally, for the convergence of a system S with degree ϵ to be decidable there must exists an algorithm D_ϵ such that $D_\epsilon(S, M_0, \delta, E) = 1$ if the system is convergent at time δ and 0 otherwise. In other words, **adaptability is decidable if**, for all M_δ, the function

$$D(M_\delta) = \begin{cases} 1 & M_\delta \text{ is adapted.} \\ 0 & \text{Otherwise.} \end{cases}$$

is computable.

If this is the case, we can describe a machine P such that given full descriptions of D_ϵ, S and M_0 it runs D_ϵ with inputs S and M_0 while running over all the possible times t, returning the first t for which the system converges. Note that

$$\delta = P(\langle D_\epsilon\rangle\langle S\rangle\langle M_0\rangle\langle E\rangle).$$

Hence we have a short description of δ and therefore δ *cannot be random*: if $S(M_0, t, E)$ is a convergent system then

$$K(\delta) \leq K(D_\epsilon) + K(S) + K(E) + K(M_0) + O(1), \tag{4.3}$$

where δ is the first time at which convergence is reached. Note that all the variables are known at the initial state of the system. This result can summed up by the following lemma:

Lemma 4.2. *Let S be a system convergent at time δ. If δ is considerably more descriptively complex than the system and the environment, i.e. if for every reasonably large natural number d we have it that*

$$K(\delta) > K(S) + K(E) + K(M_0) + d,$$

then δ cannot be found by an algorithm described within d number of characters.

Proof. It is a direct consequence of the inequality 4.3. □

We call such times *random convergence times* and the state of the system M_δ a *random state.*

It is important to note that the descriptive complexity of a random state must also be high:

Lemma 4.3. *Let S be a convergent system with a complex state $S(M_0, \delta, E)$. For every reasonably large d we have it that*

$$K(S(M_0, \delta, E)) > K(S) + K(E) + K(M_0) + d.$$

Proof. Suppose the contrary to be true, i.e. that there exists d small enough that

$$K(S(M_0, \delta, E)) \leq K(S) + K(E) + K(M_0) + d.$$

Let q be the program that, given S, E, M_0 and $S(M_0, \delta, E)$, runs $S(M_0, t, E)$ in order for each t and compares the result to $S(M_0, \delta, E)$, returning the first time where the equality is reached. Therefore, given the uniqueness of the successor state (3.1), $\delta = q(S, M_0, E, S(M_0, \delta, E))$ and

$$\begin{aligned} K(\delta) \leq & K(S) + K(E) + K(M_0) + K(S(M_0, \delta, E)) + |q| \\ \leq & K(S) + K(E) + K(M_0) + (K(S) + K(E) + K(M_0) + d) + O(1), \end{aligned}$$

which gives us a small upper bound for the random convergence time δ. □

In other words, if δ has high descriptive complexity, then there does not exists a reasonable algorithm that finds it even if we have a complete description of the system and its environment. It follows that the descriptive complexity of a computable convergent state cannot be much greater than the descriptive complexity of the system itself.

What a *reasonably large* d is has been handled so far with ambiguity, as it represents the descriptive complexity of any computable method D_ϵ. We may intend to find convergence times, which intuitively cannot be arbitrarily large. It is easy to '*cheat*' on the inequality 4.3 by including in the description of the program D_ϵ the full description of the convergence time δ, which is why we ask for *reasonable* descriptions.

Another question left to be answered is whether complex convergence times do exist for a given limit d, considering that the limits imposed by the inequality 4.3 loosen up in direct relation to the descriptive complexity of S, E and M_0.

The next result answers both questions by proving the existence of complex convergence times for a broad characterization of the size of d:

Lemma 4.4 (Existence of Random Convergence Times)**.** *Let F be a total computable function. For any ϵ there exists a system $S(M_0, t, E(t))$ such that the convergence times are $F(S, M_0, E)$-random.*

Proof. Let E and s be two natural numbers such that $K(E|s) > \epsilon$. If we assume the non existence of $F(S, M_0, E)$-random convergence times, by reduction to the Halting Problem ([74]), is easy to reach a contradiction:

Let T' be a Turing Machine, and S_t the Turing machine that emulates T for t steps with input M_0 and returns E for every time equal to or greater than the halting time, and s otherwise. Let us consider the system

$$S(M_0, t, E(t)) = S_t(\langle T\rangle\langle M_0\rangle\langle t\rangle\langle E\rangle).$$

If the convergence times are not $F(S, M_0, E)$-random, then there exists a constant c such that we can decide HP by running S' for each t that meets the inequality

$$|t| + 2\log|t| + c \leq |S'| + |\langle T\rangle\langle M_0\rangle\langle t\rangle\langle E\rangle| + F(S, M_0, E)^1,$$

which cannot be done, since HP is undecidable. □

Let us focus on what the previous lemma is saying: F can be any computable function. It can be a polynomial or exponential function with respect to the length of a given description for M_0 and E. It can also be any computable theory that we might propose for setting an upper limit to the size of an algorithm that finds convergence times given descriptions of the system's behavior, environment and initial state. In other words, for a class of dynamical systems, finding convergence times, therefore convergent states,

[1]Recall the definition 2.9.

is not decidable, even with complete information about the system and its initial state.

Finally, by the proof of lemma 4.4, adapted states can be seen as a **generalization of halting states**.

4.2 Randomness of Convergence in Dynamic Environments

So far I have limited the discussion to fixed environments. However, as observed in the physical world, the environment itself can change over time. We call such environments *dynamic environments*. In this section I will extend the previous results to cover environments that change depending on time as well as on the initial state of the system. I will also introduce a weaker convergence condition called *weak convergence* and propose a necessary (but not sufficient) condition for the computability of convergence times called *descriptive differentiability*.

We can think of an environment E as a dynamic computable system, a moving target that also changes with time and depends on the initial state M_0. In order for the system to be convergent, I propose the same criterion—there must exists δ such that $n \geq \delta$ implies

$$K(E(M_0, n)|S(M_0, n, E(M_0, n))) \leq \epsilon. \tag{4.4}$$

A system with a dynamic environment also meets the inequality 4.3 and lemmas 4.2 and 4.4 since we can describe a machine that runs both S and E for the same time t. Given that E is a moving target it is convenient to consider an *adaptation period* for the new states of E:

Definition 4.5. We say that S *converges weakly* to E if there exist an infinity of times δ_i such that

$$K(E(M_0, \delta_i)|S(M_0, \delta_i, E(M_0, \delta_i))) \leq \epsilon. \tag{4.5}$$

We will call the infinite sequence $\delta_1, \delta_2, \ldots \delta_i \ldots$ the sequence of *adaptation times.*

As a direct consequence of the inequality 4.3 and lemma 4.4 we have the following lemma:

Lemma 4.6. *Let $S(M_0, t, E(M_0, t))$ be a weakly converging system. Any decision algorithm $D_\epsilon(S, M_0, \delta_i, E)$ can only decide the first non-random time.*

As noted above, these results do not change when dynamic environments are considered. In fact, we can think of static environments as a special case of dynamic environments. However, with different targets of adaptability and convergence, it makes sense to generalize beyond the first convergence time. Also, it should be noted that specifying a convergence index adds additional information that a decision algorithm can potentially use.

Lemma 4.7. *Let $S(M_0, t, E(M_0, t))$ be a weakly converging system with an infinity of random times such that $k > j$ implies that*

$$K(\delta_k) = K(\delta_j) + \Delta K_\delta(j, k),$$

where ΔK_δ is a (not necessarily computable) function with a range confined to the positive integers. If the function $\Delta K_\delta(i, i+m)$ is unbounded with respect to i, then any decision algorithm $D_\epsilon(S, M_0, E, i)$, where i is the i-th convergence time, can only decide a finite number of i's.

Proof. Suppose that $D_\epsilon(S, M_0, E, i)$ can decide an infinite number of instances. Let us consider two times δ_i and δ_{i+m}. Note that we can describe a program that, by using D_ϵ, S, E and M_0 and i together with the distance m, finds δ_{i+m}. The next inequality follows:

$$K(\delta_{i+m}) \leq K(D_\epsilon) + K(i) + K(m) + O(1).$$

Next, note that we can describe another program that given δ_i and using D_ϵ, S, E and M_0 finds i, from which

$$K(i) \leq K(D_\epsilon) + K(\delta_i) + O(1) \text{ and } -K(\delta_i) \leq K(D_\epsilon) - K(i) + O(1).$$

Therefore:

$$\Delta K_\delta(i, i+m) = K(\delta_{i+m}) - K(\delta_i) \leq 2K(D_\epsilon) + K(m) + O(1)$$

and $\Delta K_\delta(i, i+m)$ is bounded with respect to i. □

As a direct consequence of the previous lemma, the next definition implies in computability for a sequence of times:

Definition 4.8. We will say that a sequence of times $\delta_1, \delta_2, ..., \delta_i, ...$ is *non-descriptively differentiable* if $\Delta K_\delta(m)$ is not a total function, where

$$\Delta K_\delta(m) = \max\{|\Delta K_\delta(i, i+m)| : i \in \mathbb{N}\}.$$

In other words, an infinite sequence $\delta_1, \delta_2, ...\delta_i, ...$ of natural numbers is non-descriptively differentiable if there exists a natural number m such that the difference between the descriptive complexity of the elements at distance m, given by

$$\Delta K_\delta(i, i+m) = K(\delta_{i+m}) - K(\delta_i),$$

is unbounded. And if a sequence is *non-descriptively differentiable* then the mapping $i \mapsto \delta_i$ is incomputable.

One sequence that can be easily shown to meet the definition of descriptively differentiability is the trivial sequence produced by an enumeration machine: the sequence of natural numbers. By theorem 4.1 and lemma 4.7, it follows that

If adaptation is decidable then the algorithmic descriptive complexity of the sequence of adaptation times must follow closely that of the natural numbers.

In the next chapter, I will show that such tight bound to the descriptive algorithmic complexity for any decidable sequence will tightly bound the complexity growth of any decidable sequence to that of the trivial systems, such as the sequence of natural numbers.

Chapter 5

Beyond Halting States: Open-Ended Evolution

Inequality 4.3 states that being able to predict or recognize adaptation imposes a limit to the descriptive complexity of the first adapted state. A particular case is the **halting state**, as shown in the proof of lemma 4.4. In this chapter I extend the lemma to continuously evolving systems, showing that computability of adapted times limits the complexity of adapted states beyond the first, imposing a limit to open-ended evolution for three of the complexity measures introduced in section 2.3: sophistication, coarse sophistication and busy beaver logical depth.

For a system in constant evolution converging to a dynamic environment, the lemma 4.7 imposes a limit to the growth of the descriptive complexity of a system with computable adapted states: *if growth of the descriptive complexity of a sequence of convergent times is unbounded in the sense of definition 4.8, then all but a finite number of times are undecidable.* The converse would be convenient, however it is not always true. Moreover, the next series of results shows that imposing such a limit would impede strong OEE:

Theorem 5.1. *Let S be a non-cyclical computable system with initial state M_0, E a dynamic environment, and $\delta_1, ..., \delta_i, ...$ a sequence of times such that for each δ_i there exists a* ***total function*** *p_i such that $p_i(M_{\delta_i}) = E(\delta_i)$. If the* *function $p : i \mapsto p_i$ is computable, then the function $\delta : i \mapsto \delta_i$ is computable.*

Proof. Assume that p is computable. We can describe a program D_ϵ such that, given S, M_0, δ_i and E, runs $p_{\delta_i}(M_t)$ and $E(t)$ for each time t, returning 1 if δ_i-th t is such that $p_{\delta_i}(t) = E(t)$, and 0 otherwise. Therefore the sequence of δ_i's is computable. □

The last result can be applied naturally to weakly convergent systems (4.5): the way each adapted state approaches to E is unpredictable, in other words, its *behavior* changes over different stages unpredictably. Formally:

Corollary 5.2. *Let $S(M_0, t, E(t))$ be a weakly converging system, with adapted states $M_{\delta_1}, ..., M_{\delta_i}, ...$ and $p_1, ..., p_i, ...$ its respective adapted behavior. If the mapping $\delta : i \mapsto \delta_i$ is non-descriptively differentiable then the function $p : i \mapsto p_i$ is not computable.*

Proof. It is a direct consequence of applying the theorem 5.1 to the definition of weakly converging systems. □

Implied in the proof of the last corollary is the requirement for each p_i to be a total function. Within an evolution framework, this requirement makes sense given that, in weakly convergent systems, the program p_i represents an the behavior of a digital organism, a biological theory or other computable system that uses M_{δ_i}'s information to predict the behavior of $E(\delta_i)$. If this prediction does not process its environment in finite time then is hard to argue that it represents an *adapted system* or a *useful theory*

Let us focus on the consequence of lemmas 2.13 and 2.15 and corollary 2.19. Given the relationship established between descriptive time complexity and the corresponding state of a system (theorem 4.1), these last results imply that either the complexity of the adapted states of a system (using any of the three complexity measures) grows very slowly for an infinite subsequence of times (becoming increasingly common up to a probability limit of 1 [17]) or the subsequence of adapted times is undecidable.

Theorem 5.3. *If $S(M_0, t, E(t))$ is a weakly converging system with adaptation times $\delta_1, ..., \delta_i, ...$ that exhibits strong OEE with respect to csoph and*

$depth_{bb}$, then the mapping $\delta : i \mapsto \delta_i$ is not computable. Also, there exists a constant c such that the result applies to $soph_c$.

Proof. We can see the sequence of adapted states as a function $M_{\delta_i} : i \mapsto M_{\delta_i}$. By lemmas 2.13 and 2.15 and corollary 2.19, for the three stated measures of complexity, there exist an infinite subsequence where the respective complexity is upper bounded by $O(\log \log i)$. It follows that if complexity grows faster than $O(\log \log i)$ for an infinite subsequence, then there must exist an infinity of indexes j in the bounded succession where $\gamma(j)$ grows faster than $C(M_j)$. Therefore there exist an infinity of indexes j where $C(M_j) - \gamma(j)$ is upper bounded. Finally, note that if a computable mapping $\delta : i \mapsto \delta_i$ allows growth on the order of $O(\log \log i)$, then the computable function $\delta' : i \mapsto \delta_{2^{2^i}}$ would grow faster than the stated bound. □

Now, in the absence of absolute solutions to the problem of finding adapted states in the presence of strong OEE, one might cast about for a partial solution or approximation that decides most (or at least some) of the adapted states. The following corollary shows that the problem is not even semi-computable: *any algorithm one might propose can only decide a bounded number of adapted states.*

Corollary 5.4. *If $S(M_0, t, E(t))$ is a weakly converging system with adapted states $M_1, ..., M_i, ...$ that show strong OEE, then the mapping $\delta : i \mapsto \delta_i$ is not even semi-computable.*

Proof. Note that for any subsequence of adaptation times $\delta_{j_1}, ..., \delta_{j_k}, ...$, the system must show strong OEE. Therefore, by theorem 5.3, any subsequence must also not be computable. It follows that there cannot exists an algorithm that produces an infinity of elements of the sequence, since such an algorithm would allow the creation of a computable subsequence of adaptation times. □

In short, the theorem 5.3 imposes undecidability on strong OEE and, according to theorem 5.2, the behavior and interpretation of the system evolves in an unpredictable way, establishing one path for *emergence*: *a set of rules for future states that cannot be reduced to an initial set of rules.*

Recall that for a given weakly converging dynamical system, the sequence of programs p_i represents the behavior or interpretation of each adapted state M_i. If a system exhibits strong OEE with respect to the complexity measures $soph_c$, $csoph$ or $depth_{bb}$, by corollary 5.2 and theorem 5.3 the sequence of behaviors is incomputable, and therefore irreducible to any function of the form $p : i \mapsto p_i$, even when possessing complete descriptions for the behavior of the system, its environment and its initial state. In other words, *the behavior of iterative adapted states cannot be obtained from the initial set of rules.*

Furthermore, I conjecture that we should expect the results presented in this work to hold for all *adequate* measures of complexity:

Conjecture 5.5. *Computability bounds the growing complexity rate to that of an order of the slowest growing infinite subsequence with respect to any* adequate *complexity measure C.*

One way to understand conjecture 5.5 is that *the information of future states of a system is either contained at the initial state–hence their complexity is bounded by that initial state–; or is undecidable.* This should be a consequence given that, for any computable dynamical system, the randomness induced by time cannot be avoided.

5.1 Emergence, a Formal Definition

Emergence is understood as *objects, organism and behavior that arise out of an initial conditions, set of rules or fundamental entities yet are innovative, novel and irreducible with respect to them* [60].

In the opinion of the author, there is no stronger concept for **innovation** in a dynamical system than the incomputability required for (strong) OEE. I will use this requirement to give a formal definition for *emergent behavior.*

Definition 5.6. A sequence of natural numbers $\delta_1, \delta_2, ..., \delta_i, ...$ shows *emergent behavior* if it is irreducible.

In the context of computable evolutionary systems, a sequence of adapted states —and therefore of adapted times— is emergent if its irreducible to the initial state of the system, the environment and the rules governing its evolution. And as shown in theorem 5.3, this is a necessary condition for (strong) open-ended evolution.

In summary, if $S(M_0, t, E(t))$ is a weakly converging system with adaptation times $\delta_1, ..., \delta_i, ...$ such that the evolution of the system shows strong OEE then:

- **Adaptation is undecidable**.
- Furthermore, the set of adaptation times is not even semi-computable, therefore also the set of adapted states.
- The adapted behavior of the system is **irreducible and emergent**, as there is not a computable method to describe the behavior at each adaptation stage, therefore there must be *innovation*.

On Bennett's Logical Depth

Although I conjecture that the theorem 5.3 must also hold for logical depth as defined by Bennett [8], encompassing logical depth will require a deeper understanding of the internal structure of the relationship between computable systems and computing time, beyond the time complexity stability theorem (4.1), and might be related to open fundamental problems in computer science and mathematics. As mentioned before, finding a *low* upper bound to the growth of logical depth of all computable series of natural numbers would suggest a negative answer to the question of the existence of an efficient way of generating deep strings, which Bennett relates to the $P \neq PSPACE$ problem.

It is important to address the fact that the diagonal algorithm $\chi(n, T)$ that Bennett proposes for generating deep strings might present a contradiction to our conjecture. The algorithm produces strings of length n with depth T for a significance level $n - K(T) - O(\log n)$, where $K(T)$ must be smaller than n, and n must not be *as large* (or larger) than T to avoid shallow strings.

One possible issue with this algorithm is that the significance level is not computable, and we can expect it to vary greatly with respect to $K(T)$: For large T with small $K(T)$ (such as T^{T^T}) the significance level is nearly n, which suggests that, for a *steady* significance level with respect to times T with large $K(T)$, the growth in complexity might not be stable. This issue, along with an algorithm that consistently enumerates pairs of n and Ts such that $K(T) < n << T$ for growing T's, will be explored in future work and its solution would require a formal definition of *adequate* complexity measures.

If χ might present a challenge to the conjecture 5.5 this would suggest an important difference from the three complexity measures used in this work.

Chapter 6

A System Exhibiting Strong OEE

Given the results exposed in chapter 5, it is reasonable to hesitate on the definition of OEE. ‘*Is there even a reasonable systems that meets the exposed criteria?*’ The answer is yes, there is such a system.

6.1 Chaitin’s Evolutive System

In his Metabiology program [23], the mathematician Geogory Chaitin developed a first version of a *mathematical theory of evolution* based on AIT, more precisely based in Algorithmic Probability [43]. With the aim of providing mathematical evidence for the *adequacy* of Darwinian evolution, Chaitin defined a dynamic evolutionary model that converges to its environment significantly faster than exhaustive search, being fairly close to an *intelligent* solution to a mathematical problem that requires *maximal creativity* [21, 23, 22].

In the present section I will describe this system —which I will refer to as Chaitin’s system from now on— and proceed to show that it **exhibits strong OEE**. But first I must introduce the concept of algorithmic probability.

6.1.1 A Brief Introduction to Algorithmic Probability

Let us recall the original problem of defining randomness to decide if a coin toss is fair. If we see the random pattern 01101001 emerge during the first game we might conclude the that the coin has a good chance of being fair. However, if we see the same random pattern

$$01101001, 01101001, \ldots, 01101001$$

being repeated time after time then AIT tells us that, by considering the concatenated string or the set, the larger sequence is not longer random.

There is another way of reaching the same conclusion, we intuitively know that the **probability** of obtaining the same random sequence over consecutive throws is low. In other words, the probability of obtaining an specific random sequence must be low.

The concept of **algorithmic probability** (also known as Levin's semi-measure [52]) considers the production of randomly chosen programs that produce an output. The algorithmic probability of a string s is thus a measure that estimates the probability of a random program p producing a string s when run on a universal (prefix-free) Turing machine U.

Definition 6.1. The *algorithmic probability of a natural number s* for the prefix-free Turing machine U is defined as:

$$m_U(s) = \sum_{p:U(p)=s \text{ and } p \in \mathbb{V}} \frac{1}{2^{|p|}}, \tag{6.1}$$

where $\mathbb{V}$ is the set of all valid programs for U. In other words, the algorithmic probability of a natural number s is the probability for a randomly chosen program for an universal Turing machine to produce s.

The function $m_U(s)$ is a probability semi-measure given that, by **Kraft inequality** [49, 56], for any prefix-free set of finite binary strings $\mathbb{V}$ we have

$$\sum_{p \in \mathbb{V}} \frac{1}{2^{|p|}} \leq 1,$$

and the sum may not reach 1 due the non-halting programs.

A fundamental result in algorithmic probability [12] states that

$$-\log(m(s)) = K(s) + O(1). \tag{6.2}$$

Intuitively, the last equation is true given that the largest contributor to $m(s)$ is $1/2^{|s^*|}$, where $|s^*|$ is the longitude of the shortest program that generates s, and $-\log(1/2^{|s^*|}) = |s^*|$. Formally:

Theorem 6.2. *The Coding Theorem [70, 52]: For $s \in \mathbb{N}$ and an universal prefix-free Turing machine U there exists c_u such that*

$$|-\log(m(s)) - K(s)| < c_u, \tag{6.3}$$

where c_u is a fixed constant, independent of s but dependent on U.

Now, we can rewrite the equation 6.2 as

$$m_U(s) = \frac{1}{2^{K(s)+O(1)}} \tag{6.4}$$

and, by the invariance theorem (2.8), it follows that for any two prefix-free universal Turing machines U_1 and U_2 and natural number s there exists c such that

$$m_{U_1}(s) \leq c \cdot m_{U_2}(s).$$

In other words, algorithmic probability has its own version of invariance.

Through the algorithmic probability literature, the reference universal machine U is omitted, and the resulting probability $m(s)$ function is called the *Universal Distribution* [46, 69].

6.1.2 Evolution Dynamics

In Chaitin's system, the environment is defined as two *information-content equivalent* incomputable objects: the busy beaver function (definition 2.18) and Chaitin's constant Ω [15]. These two object are equivalent from a computability point of view as it is possible to obtain one from the other with a fixed program [32], therefore they convey the *same amount of (infinite) information.* Formally, Ω is defined as:

Definition 6.3. For a prefix-free universal machine U, *Chaitin's constant* Ω is defined as the probability

$$\Omega_U = \sum_{p \in HP} \frac{1}{2^P}.$$

In other words, the probability of a randomly chosen valid program P to halt for U, also known as the *halting probability* of U.

The exact value of Ω_U depends on U. However, since Ω_U encodes the busy beaver function, given the exact value for Ω_U is possible to find Ω'_U for any other universal prefix-free machine U' using the corresponding compiler and a fixed program, therefore the subindex U is omitted for the *universal constant* Ω. Let us define the **environment** at time t as $E(t) = \Omega_t$, that is the first t bits of Ω.

Now, the initial state of the system M_0 is defined as the natural number 0, the empty string. A *successful mutation* at the time t is defined as the program μ_t such that $0.M_{t-1} < 0.\mu_t(M_{t-1}) < \Omega$, where $0.X$ is the binary sequence X interpreted as a binary expansion of a number between 0 and 1. We can now define the evolution function as

$$M_t = S(M_{t-1}, t, \Omega_t) = \mu_t(M_{t-1}).$$

Now, the question is how to find μ_t. There are several strategies we can use. One is to simply try all the possible programs until we find a successful mutation. Another is to randomly choose a program according to it's classical probability. Other strategies were suggested by Chaitin: *intelligently* choose the mutations and to randomly chose a program according to the probability given by the **universal distribution** 6.1.

When using the universal distribution, Chaitin proved that the expected numbers of *stochastic draws* (or real computing time) needed to find the sequence $\mu_0, \mu_1, ..., \mu_t$ is in the order of $O(t^2(\log(t))^{1+O(1)})$ [23, pp. 127], which is much faster than going through all the programs or choosing n-bit mutations *according to classical probability*, whose time is expected to be exponential. Drawing mutations from the universal distribution yields a time that is fairly close to the linear time expected from an intelligent solution. Instead of repeating Chaitin's work here, I will keep the deterministic approach to reach a related result based on similar ideas.

Theorem 6.4. *Let us sort all possible programs according to their probability under the universal distribution, from the most probable to the least one. If S is a program that applies all the mutations according to this order then it will produce the list $\mu_0, \mu_1, ..., \mu_t$ of successful mutations in a real computing time of $O(t^2)$.*

Proof. Consider the mutations μ_k^* which are programs that add $1/2^k$ to the present state (approximation to Ω) and do the following (where M is the input):

- Run all the valid programs from length 0 to k in dovetailing fashion, adding $1/2^j$ at each successful halt until obtaining $0.M + 1/2^k$.

Note that if $0.M + 1/2^k > \Omega$, then $\mu_k^*(M)$ will never halt. Now, the algorithmic complexity of each μ_k^* is $O(\log(k))$, therefore there are at most $O(2^{\log(k)})$ valid programs before it in the ordered list. It follows that, for each state of the evolution, the program S will take a computing time of $O(t)$ and, for the complete list, $\sum_{j=0}^{t} O(j) = O(t^2)$. □

Now, we have the complete deterministic evolutionary system $S(0, t, \Omega_t)$, which runs with quadratic overhead. However, the system is not computable, as S needs to *know* when the failed mutations μ_k^* —along with other valid programs— will halt, and the program order (the universal distribution) along with the environment itself are incomputable entities.

6.2 Chaitin's System and OEE

Given that Ω can be used to compute $BB(n)$ [32], an equivalent, but much slower deterministic version of Chaitin's system is the following:

$$\begin{aligned} &M_0 = 0 \\ &M_t = p.T(p) = \max\{T(U'(q)) : H(M_{t-1}, q) \leq w\}, \end{aligned} \tag{6.5}$$

where $H(M_{t-1}, p)$ is the *distance* between the programs M_{t-1}, q is the quantification of the number of mutations needed to transform one string into the other, and w is a positive integer acting as an accumulator that resets to 1 whenever M_t increases in value, adding 1 otherwise.

The environment can be defined as either $E(t) = BB(t)$. or by an encoding of the proposition *larger than* $U(M_{t-1})$ for each time t. Given that we can compute M_{t-1} and its relationship with M_t given a description of the latter and a constant amount of information (ϵ), we find adaptation at the times t where the busy beaver function grows. It is easy to see that the sequence of programs $i \mapsto M_i$ is precisely what generates the busy beaver sequence $\eta_i = BB(i)$.

As expected, the system is not computable. $BB(t)$ is not a computable function, the evolution of the system, along with the respective adaptation times are also not computable. However, this sequence is composed of programs that compute, in order, an element of a sequence that exhibits strong OEE with respect to $depth_{bb}$:

Theorem 6.5. *The sequence of busy beaver values $\eta_i = BB(i)$ show strong OEE with respect to the complexity measure $depth_{bb}$ (definition 2.17).*

Proof. Let $\eta_i = BB(i)$ be the sequence of all busy beaver values; by definition, if i is the first value for which $BB(i)$ was obtained,

$$depth_{bb}(BB(i)) = \min\{|p_i| - K(BB(i)) + i\},$$

where $U(P_i) = K(BB(i))$.

It follows that $K(BB(i)) = |p_i|$ and $depth_{bb}(BB(i)) = i$, otherwise p_i would not be the minimal program. Therefore, the complexity of the sequence is a monotone growing function with respect of the index, hence it exhibits strong OEE with respect to $depth_{bb}$. □

And, by corollary 2.19, the next result follows.

Corollary 6.6. *The system described in equation 6.5 exhibits strong OEE with respect to csoph (definition 2.14).*

Proof. By theorems 5.2 at [4] and 6.5, for any state of the system M_i we have that $|csoph(M_i) - i| \leq O(\log(|M_i|))$, $|M_i| = K(BB(i))$ and $|M_i| \leq K(M_i)$. It follows that

$$i - O(\log(K(BB(i))) \leq csoph(M_i),$$

which, by definition of BB (2.18), implies that

$$i - O(\log(i)) \leq csoph(M_i).$$

Thus, there exists j such that $i > j$ implies that $i \mapsto csoph(M_i)$ grows monotonically, therefore and we have the result. □

Now, the corollary 6.6 proves that Chaitin's system **exhibits strong OEE** with respect at least one of the universal complexity measure used through this text.

6.3 Evolution as ¬HP

Computing the system described in section 6.1 requires a solution for the Halting Problem, and the system itself might also seem *unnatural* at first glance, as it based on abstract, theoretical mathematical constructs. However, we can think of the biosphere as a huge parallel computer that is constantly approximating solutions to the adaptation problem by means of survivability, just as Ω has been approximated [14].

I claim that *just as we cannot generally know whether a Turing machine will halt until it does, we may not know if an organism will keep adapting and survive in the future, but we can know when it failed to do so (extinction).*

One may consider biological evolution to be a very rough, but fundamental, analogue to Chaitin's system as the pursue of the busy beaver values (longest running TMs) is equivalent to the search for the longest surviving organisms. Within this framework, the periods of increase in complexity are a natural consequence of approximating BB. In this view, our assertion that adaptation is a generalization of the Halting problem has a natural interpretation. In summary, I state that:

- The Biosphere is a massive parallel computer constantly approximating the algorithmic probability Ω:
  - ϵ-adapted is a generalization of the **non-halting condition**.
  - Evolution looks for longest surviving organisms, analogue to longest running Turing machine (the busy beaver problem, which is equivalent to computing Ω).
  - The observed increase in complexity is a natural consequence of this search.
  - Simple, long surviving organisms can bee seen as rough equivalent to simple non-stopping Turing machines.

The set ¬HP is defined as the set of non-halting or diverging valid programs. This set is not semi-computable: If we had a computable function d that returned 1 on positive instances, by running it along with any program, we could decide HP. As corollary 5.4 shows, this is the same property that we find for strong OEE evolutionary systems, which is congruent with the arguments stated above.

Simply put, *deciding adaptation is equivalent to deciding the set membership of ¬HP.*

6.4 Algorithmic Probability Evolution

One possible reading for the undecidability results exposed in chapter 5 is that chasing for strong OEE is a scientific dead-end, as the sequence of adapted states are not even-semi computable. However, the beautiful *algorithmic evolution and algorithmic probability theory* advanced by Chaitin, Kolmogorov, Levin and Solomonoff presented in this chapter exposed a strong OEE system and opened a new path for studying evolution.

Definition 6.7. An evolutionary system is *driven by algorithmic probability* if the chance of a given mutation to occur is given by the universal distribution. We call these mutations *algorithmic probable mutations.*

The objective for the remainder of this work is to show a first numerical experiment in evolution using algorithmic probability as main driver of mutation, comparing the obtained results with the alternative hypothesis that mutations are *uniformly random.* For that, we first need an effective approximation to the universal distribution, which will be presented in the next chapter.

Chapter 7

Approximating the Universal Distribution

As stated before, the universal distribution $m(s)$ is an incomputable function. However, by approximating the algorithmic complexity $K(s)$, $m(s)$ can be approximated with different degrees of success (see equation 6.4). Conversely, $K(s)$ can be approximated from $m(s)$. This last approach is known as the *Coding Theorem Method* (CTM) [52]. Formally:

$$K(s) = -\log(m(s)) + O(1). \tag{7.1}$$

In the present chapter I will present an specific method, called the **Block Decomposition Method (BDM)**, developed by Zenil et al. [83, 84, 86] for approximating the descriptive complexity of a multidimensional finite objects. In one of the cited articles, we compared it to widely used alternative methods such as Shannon's entropy and compression algorithms. I will also present such comparisons.

BDM has been successfully applied to graph theory [84], image classification [34] and human behavioral complexity in the last few years [33, 44].

The content of the following sections overlap with the article ***'A Decomposition Method for Global Evaluation of Shannon Entropy and***

Local Estimations of Algorithmic Complexity' [86], for which I was a contributing author.

7.1 Approximating K

In this section, based on [86], we present a measure of algorithmic complexity that lies half-way between two universally used measures that enables the action of both at different scales by dividing data into smaller pieces for which the halting problem involved in an uncomputable function can be partially circumvented, in exchange for a huge calculation based upon the concept of algorithmic probability. The calculation can, however, be precomputed and reused in future applications, thereby constituting a strategy for efficient estimations –bounded by Shannon entropy and by algorithmic complexity– in exchange for a loss of accuracy.

7.1.1 The Use and Misuse of Lossless Compression

Lossless compression algorithms have dominated the landscape of computable applications of algorithmic complexity and have traditionally been used to approximate K. When researchers have chosen to use lossless compression algorithms for reasonably long strings, the method has proven to be of value (e.g. [25]). Their successful application has to do with the fact that compression is a sufficient test for algorithmic non-randomness (though the converse is not true). However, implementations of lossless compression algorithms are based upon estimations of *entropy* rate, and are therefore no more closely related to algorithmic complexity than is Shannon entropy by itself.

Entropy for information theory was originally conceived by Shannon [66] as a measure of information transmitted over an stochastic communication channel with known alphabets and is defined as:

Definition 7.1. The entropy H of a discrete random variable s with (finite) possible values $s_1, \ldots, s_n$ with probability distribution $P(s)$ is defined as:

$$H(s) = -\sum_{i=1}^{n} P(s_i)\log(P(s_i)),$$

where in the case of $P(s_i) = 0$ for some i, the value of the corresponding summand 0 log(0) is taken to be 0.

We can also consider blocks of symbols for higher order entropy. The following function H_l gives what is variously denominated as block entropy and is Shannon entropy over blocks or subsequence of s of length l. That is,

Definition 7.2.

$$H_l(s) = -\sum_{b \in blocks} P_l(b)\log(P_l(b)),$$

where *blocks* is the set resulting of decomposing s in substrings or blocks of size l and $P_l(b)$ is the probability of obtaining the combination of n symbols corresponding to the block b.

For infinite strings assumed to originate from a stationary source, the *entropy rate* of s can be defined as the limit

$$\lim_{l \to \infty} \frac{1}{l} \sum_{|s'|=l} H_l(s'),$$

where $|s'| = l$ indicates we are considering all the generated strings of length l. For a fixed string we can think on the normalized block entropy value where l better captures the periodicity of s.

When the logarithm is base 2, Shannon's entropy measures the average number of bits it would take transmit the full message over a communication channel. This establishes hard limits to maximum lossless compression rates. For instance, the Shannon coding (and Shannon-Fano) sorts the symbols of an alphabet according to their probabilities, assigning smaller binary self-delimited sequences to symbols that appear more frequently. Such methods form the base of many, if not most, commonly used compression algorithms.

Given its utility in data compression, entropy is often used as a measure of the information contained in a finite string $s = s_1 s_2 \ldots x \ldots s_k$. Let's consider the *natural distribution*, the uniform distribution that makes the least number of assumptions but does consider every possibility equally likely and is thus uniform. Suggested by the set of symbols in s and the string length the *natural distribution* of s is the distribution defined by $P(x) = \frac{n_x}{|s|}$, where n_x is the number of times the object x occurs in s (at least one to be considered), and the respective entropy function H_l. If we consider blocks of size $n >> l \geq 2$ and the string $s = 01010101\ldots01$, where n is the length of the string, then s can be compressed in a considerably smaller number of bits than a statistically random sequence of the same length and, correspondingly, has a lower H_l value.

When the communication channel is transmitting an infinite string s, the *entropy rate* of s is defined as the limit

$$\lim_{l\to\infty} H_l(s).$$

For finite objects, we can think of it as the normalized block entropy value where l *better captures* the periodicity of s. It follows that, depending on the granularity and probability distribution used, **entropy can only account for statistical regularities and not for algorithmic ones**. For example, the string 01010101 is periodic, but for the smallest granularity (1 bit) or 1-symbol block, the sequence has maximal entropy, because the number of 0s and 1s is the same assuming a uniform probability distribution for all strings of the same finite length. Only for longer blocks of length 2 bits can the string be found to be regular, identifying the smallest entropy value for which the granularity is at its minimum.

Now, **data compression** can be viewed as a function that maps data onto other data using the same units or alphabet (if the translation is into different units or a larger or smaller alphabet, then the process is called an encoding). Compression is successful if the resulting data is shorter than the original data plus the decompression instructions needed to fully reconstruct said original data. For a compression algorithm to be lossless, there must be a reverse mapping from compressed data to the original data. That is to say, the compression method must encapsulate a bijection between "plain" and "compressed" data, because the original data and the compressed data should be in the same units.

In similar fashion to Shannon-Fado code, the classical LZ77 [87] and LZ78

[88] compression algorithms enact a greedy parsing of the input data LZ77 and achieve compression by replacing repeated chunks of data with references to a single copy of that data existing earlier in the uncompressed object. That is, at each step, they take the longest dictionary phrase which is a prefix of the currently unparsed string suffix. LZ algorithms are said to be 'universal' because, assuming unbounded memory (arbitrary sliding window length), they asymptotically approximate the (infinite) entropy rate of the generating source [88]. Not only does lossless compression fail to provide any estimation of the algorithmic complexity of small objects [29, 68], it is also not more closely related to algorithmic complexity than Shannon entropy by itself, being only capable of exploiting statistical regularities (if the observer has no other method to update/infer the probability distribution). Thus in effect no general lossless compression algorithm does better than provide the Shannon entropy rate of the objects it compresses.

Another immediate drawback of computable entropy based complexity measures is the lack of robustness [80, 78]. In other words, they are not invariant to different descriptions of the same object–unlike algorithmic complexity, where the invariance theorem guarantees the invariance of an object's algorithmic complexity.

BDM builds upon block entropy's decomposition approach using algorithmic complexity methods to obtain and combine its building blocks. The result is a complexity measure that, as shown in section 7.2.2, approaches K in the best case and behaves like entropy in the worst case (7.2.5), outperforming H_l in various scenarios. First we introduce the algorithm that conforms the building blocks of BDM, which are local estimations of algorithmic complexity.

7.1.2 The Coding Theorem Method (CTM)

A computationally expensive procedure that is nevertheless closely related to algorithmic complexity involves approximating the algorithmic complexity of an object by running every possible program, from the shortest to the longest, and counting the number of times that a program produces every string object. The length of the computer program will be an upper bound of the algorithmic complexity of the object, following the Coding theorem 6.2, and a (potentially) compressed version of the data itself (the shortest program found) for a given computer language or 'reference' UTM. This guarantees discovery of the shortest program in the given language or reference UTM but entails an exhaustive search over the set of countable infinite computer programs that can be written in such a language. A program of length n has asymptotic probability close to 1 of halting in time 2^n [17], making this procedure exponentially expensive, even assuming that all programs halt or that programs are assumed never to halt after a specified time, with those that do not being discarded.

As shown in [29] and [68], an exhaustive search can be carried out for a small-enough number of computer programs (more specifically, Turing machines) for which the halting problem is known, thanks to the Busy Beaver game [62]. This is the challenge of finding the Turing machine of fixed size that runs for longer than any other machine of the same size, and for which values are known for Turing machines with 2 symbols and up to 4 states. One can also proceed with an informed runtime cut-off, well below the theoretical 2^n optimal runtime that guarantees an asymptotic drop of non-halting machines [17].

The so called *Coding Theorem Method* (or simply CTM) is a Bottom-up Approach to Algorithmic Complexity. Unlike common implementations of lossless compression algorithms, the main motivation of CTM is to find algorithmic signals rather than just statistical regularities in data, beyond the range of application of Shannon entropy and entropy rate.

CTM is divided into two parts thanks to the incomputability and intractability intrinsic to algorithmic complexity measures. The trade-off of CTM is this two-step and two-speed algorithm that requires a calculation in $O(exp)$ time but that can then be used and applied in $O(1)$ by exchanging time for memory in the construction of a precomputed lookup table, which in turn diminishes the precision of the method in proportion to the object size (e.g. string length) unless a new $O(exp)$ iteration is precomputed. This means that the procedure only works well for short strings. The slow part of the algorithm is, paradoxically, slower than the slowest computable function (equivalent to calculating the busy beaver function ([62], see definition 2.18), but some shortcuts and optimizations are possible [29, 68].

CTM is rooted in the relation [29, 68] provided by algorithmic probability between frequency of production of a string from a random program and its algorithmic complexity as described by equation (7.1). Essentially it uses the fact that the more frequent a string is, the lower Kolmogorov complexity it has; and strings of lower frequency have higher Kolmogorov complexity. The advantage of using algorithmic probability to approximate K by application of the Coding Theorem6.2 is that $m(s)$ produces reasonable approximations to K based on an average frequency of production, which retrieves values even for small objects.

Let (t, k) denote the set of all Turing machines with t states and k symbols using the Busy Beaver formalism [62], and let T be a Turing machine in (t, k) with empty input. Then the empirical output distribution $D(t, k)$ for a sequence s produced by some $T \in (t, k)$ gives an estimation of the *algorithmic probability* of s, $D(t, k)(s)$ defined by:

Definition 7.3.

$$D(t,k)(s) = \frac{|\{T \in (t,k) : T \ \textit{produces } s\}|}{|\{T \in (t,k) : T \ \textit{halts}\ \}|} \tag{7.2}$$

For small values t and k, $D(t, k)$ is computable for values of the Busy Beaver problem that are known. In this context, the Busy Beaver problem is the problem of finding the t-state, k-symbol Turing machine which writes a maximum number of non-blank symbols before halting, starting from an empty tape, or the Turing machine that performs a maximum number of steps before halting, having started on an initially blank tape. For $t = 4$ and $k = 2$, for example, the Busy Beaver machine has maximum runtime $S(t) = 107$ [11], from which one can deduce that if a Turing machine with 4 states and 2 symbols running on a blank tape hasn't halted after 107 steps, then it will never halt. This is how D was initially calculated– by using known Busy Beaver values. However, because of the undecidability of the Halting problem, the Busy Beaver problem is only computable for small t, k values [62]. Nevertheless, one can continue approximating D for a greater number of states (and alphabet size), proceeding by sampling, as described in [29, 68], with an informed runtime based on both theoretical and numerical results.

Notice that $0 < D(t, k)(s) < 1$, $D(t, k)(s)$ and is thus said to be a semi-measure, just as $m(s)$ is.

Now we can introduce a measure of complexity that is heavily reliant upon *algorithmic probability* $m(s)$, as follows:

Definition 7.4. Let (t, k) be the space of all t-state k-symbol Turing machines, $t, k > 1$ and $D(t, k)(s) =$ the function assigned to every finite binary string s. Then:

$$CTM(s, t, k) = -\log(D(t, k)(s)). \tag{7.3}$$

That is, the more frequently a string is produced the lower its Kolmogorov complexity, with the converse also being true.

Table 7.1 shows the rule spaces of Turing machines that were explored, from which empirical algorithmic probability distributions were sampled and estimated.

(t,k)	**Calculation**	**Number of Machines**	**Time**
(2,2)	$F-$ (6 steps)	$\lvert R(2,2)\rvert = 2000$	0.01 s
(3,2)	$F-$ (21)	$\lvert R(3,2)\rvert = 2\,151\,296$	8 s
(4,2)	$F-$ (107)	$\lvert R(4,2)\rvert = 3\,673\,320\,192$	4 h
$(4,2)_{2D}$	$F_{2D}-$ (1500)	$\lvert R(4,2)_{2D}\rvert = 315\,140\,100\,864$	252 d
(4,4)	S (2000)	334×10^9	62 d
(4,5)	S (2000)	214×10^9	44 d
(4,6)	S (2000)	180×10^9	41 d
(4,9)	S (4000)	200×10^9	75 d
(4,10)	S (4000)	201×10^9	87 d
(5,2)	$F-$ (500)	$\lvert R(5,2)\rvert = 9\,658\,153\,742\,336$	450 d
$(5,2)_{2D}$	S_{2D} (2000)	1291×10^9	1970 d

Table 7.1: Calculated empirical distributions from rulespace (t, k). Letter codes: F full space, S sample, $R(t, k)$ reduced enumeration. Time is given in seconds (s), hours (h) and days (d).

Definition 7.5. We will designate as *base string*, *base matrix*, or *base tensor* the objects of size l for which CTM values were calculated such that the full set of k^l objects have CTM evaluations. In other words, the base object is the maximum granularity of application of CTM.

Table 7.1 provides figures relating to the number of base objects calculated.

Validations of CTM undertaken before show the correspondence between CTM values and the exact number of instructions used by Turing machines when running to calculate CTM [67](Fig 1 and Table 1) to produce each string, i.e. direct K complexity values for this model of computation (as opposed to CTM using algorithmic probability and the Coding theorem) under the chosen model of computation [62]. The correspondence in values found between the directly calculated K and CTM by way of frequency production was near perfect.

Sections 7.1.2 and 7.2 and Fig. 10, 11, 12 and 15 in [36] support the agreements in correlation using different rule spaces of Turing machines and different computing models altogether (cellular automata). Section 7.1.1 of the same paper provides a first comparison to lossless compression. The sections 'Agreement in probability' and 'Agreement in rank' provide further material comparing rule space (5,2) to the rule space (4,3) previously calculated in Zenil and Delahaye [29]. The section 'Robustness' in Soler-Toscano et al. [68] provides evidence relating to the behavior of the *invariance theorem constant* for a standard model of Turing machines [62].

7.2 The Block Decomposition Method (BDM)

Because finding the program that reproduces a large object is computationally very expensive and ultimately uncomputable, one can aim at finding smaller programs that reproduce smaller parts of the original object, parts that together compose the larger object. And this is what the BDM does. It decomposes the original data into fragments, the Kolmogorov complexity of which we have good approximations of. The sum of the programs that reconstruct the original data is an approximation of its Kolmogorov complexity. This also means that the method is local. BDM cannot, however, work alone. It requires a CTM, described in the next section. In Section 7.5 we study a method to extend its range of application, thereby effectively extending its power by combining it with Shannon entropy.

The BDM is a hybrid complexity measure that calculates global entropic and local algorithmic complexity. It is meant to extend the power of CTM, and consists in decomposing objects into smaller pieces for which exact complexity approximations have been numerically estimated using CTM, then reconstructing an approximation of the Kolmogorov complexity for the larger object by adding the complexity of the individual components of the object, according to the rules of information theory. For example, if s is an object and $10s$ is a repetition of s ten times smaller, upper bounds can be achieved by approximating $K(s) + \log_2(10)$ rather than $K(10s)$, because we know that repetitions have a very low Kolmogorov complexity, given that one can describe repetitions with a short algorithm.

Here we introduce and study the properties of this *Block Decomposition Method* based on a method advanced in [29, 68] that takes advantage of the powerful relationship established by algorithmic probability between the frequency of a string produced by a random program running on a (*prefix-free*) UTM and the string's Kolmogorov complexity. The chief advantage of this method is that it deals with small objects with ease, and it has shown stability in the face of changes of formalism, producing reasonable Kolmogorov complexity approximations. BDM must be combined with CTM if it is to scale up properly and behave optimally for upper bounded estimations of K. BDM + CTM is universal in the sense that it is guaranteed to converge to K due to the invariance theorem, and as we will prove later, if CTM no longer runs, then BDM alone approximates the Shannon entropy of a finite object.

Like compression algorithms, BDM is subject to a trade-off. Compression algorithms deal with the trade-off of compression power and compression/decompression speed.

7.2.1 *l*-overlapping String Block Decomposition

Let us fix values for t and k and let $D(t,k)$ be the frequency distribution constructed from running all the Turing machines with n states and k symbols. Following Eq. (7.2), we have it that $-\log D$ is an approximation of K (denoted by CTM). We define the BDM of a string or finite sequence s as follows,

Definition 7.6.

$$BDM(s,l,m) = \sum_i CTM(s^i,m,k) + \log(n_i) \tag{7.4}$$

where n_i is the multiplicity of s^i and s^i the subsequence i after decomposition of s into subsequences s^i, each of length l, with a possible remainder sequence $y < |l|$ if $|s|$ is not a multiple of the decomposition length l.

The parameter m goes from 1 to the maximum string length produced by CTM, where $m = l$ means no overlapping inducing a partition of the decomposition of s, m is thus an overlapping parameter when $m < l$ for which we will investigate its impact on BDM (in general, the smaller m a greater overestimation of BDM).

The parameter m is needed because of the remainder. If $|s|$ is not a multiple of the decomposition length l then the option is to either ignore the remainder in the calculation of BDM or define a sliding window with overlapping $m - l$.

The choice of t and k for CTM in BDM depend only on the available resources for running CTM, which involves running the entire (t,k) space of Turing machines with t symbols and k states.

BDM approximates K in the following way: if p_i is the minimum program that generates each base string s^i, then $CTM(s^i) \approx |p_i|$ and we can define an unique program q that runs each p_i, obtaining all the building blocks. How many times each block is present in s can be given in $O(\log(n_i))$ bits.

Therefore, BDM is the sum of the information needed to describe the decomposition of s in base strings. How close is this sum to K is explored in section 7.2.2.

The definition of BDM is interesting because one can plug in other algorithmic distributions, even computable ones, approximating some measure of algorithmic complexity even if it is not the one defined by Kolmogorov-Chaitin such as, for example, the one defined by Calude et al. [13] based upon Finite-state automata. BDM thus allows the combination of measures of classical information theory and algorithmic complexity.

For example, for binary strings we can use $t = 2$ and $k = 2$ to produce the empirical output distribution $(2, 2)$ of all machines with 2 symbols and 2 states by which all strings of size $l = 12$ are produced, except two (one string and its complement). But we assign them values

$$\max \{CTM(y, 2, 2) + e : |y| = 12\}$$

where e is different from zero because the missing strings were not generated in $(2, 2)$ and therefore have a greater algorithmic random complexity than any other string produced in $(2, 2)$ of the same length. Then, for $l = 12$ and $m = 1$, $BDM(s, l, m)$ decomposes $s = 010101010101010101$ of length $|s| = 18$ into the following subsequences:

010101010101
101010101010
010101010101
101010101010
010101010101
101010101010
010101010101

with 010101010101 having multiplicity 4 and 101010101010 multiplicity 3.

We then get the CTM values for these sequences:

$$CTM(010101010101, 2, 2) = 26.99073$$
$$CTM(101010101010, 2, 2) = 26.99073$$

To calculate BDM, we then take the sum of the CTM values plus the sum of the $\log_b$ of the multiplicities, with $b = 2$ because the string alphabet is 2, the same as the number of symbols in the set of Turing machines producing the strings. Thus:

$$\log_2(3) + \log_2(4) + 26.99073 + 26.99073 = 57.56642$$

7.2.2 2- and w-Dimensional Complexity

To ask after the likelihood of an array, we can consider a 2-dimensional Turing machine. The Block Decomposition Method can then be extended to objects beyond the unidimensionality of strings, e.g. arrays representing bitmaps such as images, or graphs (by way of their adjacency matrices). We would first need CTM values for 2- and w-dimensional objects that we call base objects (e.g. base strings or base matrices).

A popular example of a 2-dimensional tape Turing machine is Langton's ant [50]. Another way to see this approach is to take the BDM as a way of deploying all possible 2-dimensional deterministic Turing machines of a small size in order to reconstruct the adjacency matrix of a graph from scratch (or smaller pieces that fully reconstruct it). Then, as with the *Coding theorem method* (above), the algorithmic complexity of the adjacency matrix of the graph can be estimated via the frequency with which it is produced from running random programs on the (prefix-free) 2-dimensional Turing machine. More specifically,

$$BDM(X, \{x_i\}) = \sum_{(r_i, n_i) \in Adj(X)_{\{x_i\}}} CTM(r_i) + \log(n_i), \tag{7.5}$$

where the set $Adj(X)_{\{x_i\}}$ is composed of the pairs (r, n), r is an element of the decomposition of X (as specified by a *partition* $\{x_i\}$, where x_i is a submatrix of X) in different sub-arrays of size up to $d_1 \times \ldots \times d_w$ (where w is the dimension of the object) that we call *base matrix* (because CTM values were obtained for them) and n is the multiplicity of each component. $CTM(r)$ is a computable approximation from below to the algorithmic information

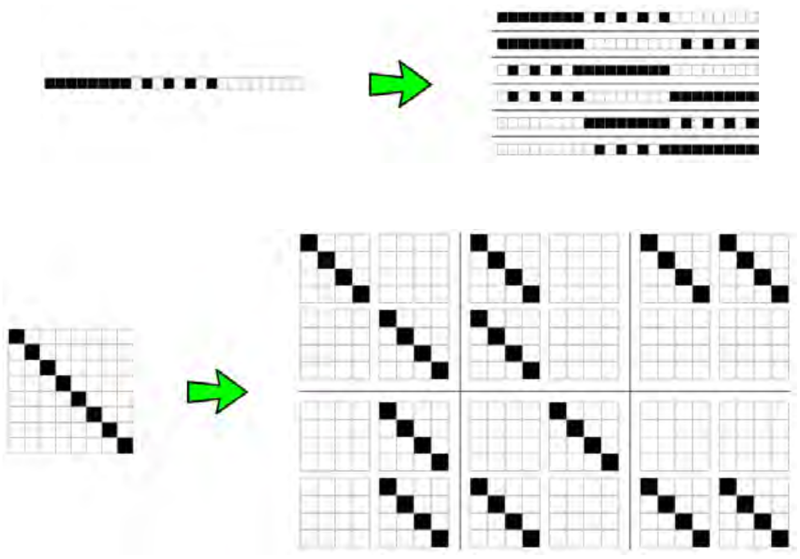

Figure 7.1: Non-overlapping BDM calculations are invariant to block permutations (reshuffling base strings and matrices), even when these permutations may have different complexities due to the reorganization of the blocks that can produce statistical or algorithmic patterns. For example, starting from a string of size 24 (top) or an array of size 8×8 (bottom), with decomposition length $l = 8$ for strings and decomposition $l = 4 \times 4$ block size for the array, all 6 permutations for the string and all 6 permutations for the array have the same BDM value regardless of the shuffling procedure.

complexity of r, $K(r)$, as obtained by applying the coding theorem method to w-dimensional Turing machines. In other words, $\{r_i\}$ is the set of *base objects*.

Because string block decomposition is a special case of matrix block decomposition, and square matrix block decomposition is a special case of w-block decomposition for objects of w dimensions, let us describe the way in which BDM deals with boundaries on square matrices, for which we can assume CTM values are known, and that we call base strings or base matrices.

Fig. 7.1 shows that the number of permutations is a function of the complexity of the original object, with the number of permutations growing in proportion to the original object's entropy–because the number of different resulting blocks determines the number of different n objects to distribute among the size of the original object (e.g. 3 among 3 in Fig. 7.1 (top) or only 2 different 4×4 blocks in Fig. 7.1 (bottom)). This means that the non-overlapping version of BDM is not invariant vis-à-vis the variation of the entropy of the object, on account of which it has a different impact on the error introduced in the estimation of the algorithmic complexity of the object. Thus, non-overlapping objects of low complexity will have little impact, but with random objects non-overlapping increases inaccuracy. Overlapping decomposition solves this particular permutation issue by decreasing the number of possible permutations, in order to avoid trivial assignment of the same BDM values. However, overlapping has the undesired effect of systematically overestimating values of algorithmic complexity by counting almost every object of size n, $n-1$ times, hence overestimating at a rate of about $n(n-1)$ for high complexity objects of which the block multiplicity will be low, and by $n\log(n)$ for low complexity objects.

Applications to graph theory [84], image classification [34] and human behavioral complexity have been produced in the last few years [33, 44].

BDM Upper and Lower Absolute Bounds

In what follows we show the hybrid nature of the measure. We do this by setting lower and upper bounds to BDM in terms of the algorithmic complexity $K(X)$, the partition size and the approximation error of CTM, such that these bounds are tighter in direct relation to smaller partitions and more accurate approximations of K. These bounds are independent of the partition strategy defined by $\{x_i\}$.

Proposition 7.7. *Let BDM be the function defined in Eq. 7.5 and let X be an array of dimension w. Then*

$$K(X) \leq BDM(X, \{x_i\}) + O(\log^2 |A|) + \epsilon$$

and

$$BDM(X, \{x_i\}) \leq |Adj(X)_{\{x_i\}}| K(X) + O(|Adj(X)_{\{x_i\}}| \log |Adj(X)_{\{x_i\}}|) - \epsilon,$$

where A is a set composed of all possible ways of accommodating the elements of $Adj(X)_{\{x_i\}}$ in an array of dimension w, and ϵ is the sum of errors for the approximation CTM over all the sub-arrays used.

Proof. Let $Adj(X)_{\{x_i\}} = \{(r_1, n_1), ..., (r_k, n_k)\}$ and $\{p_j\}$, $\{t_j\}$ be the sequences of programs for the reference prefix-free UTM U such that, for each $(r_j, n_j) \in Adj(X)_{\{x_i\}}$, we have $U(p_j) = r_j$, $U(t_j) = n_j$, $K(r_j) = |p_j|$ and $|t_j| \leq 2\log(n_j) + c$. Let ϵ_j be a positive constant such that $CTM(r_j) + \epsilon_j = K(X)$; this is the error for each sub-array. Let ϵ be the sum of all the errors.

For the first inequality we can construct a program q_w, whose description only depends on w, such that, given a description of the set $Adj(X)_{\{x_i\}}$ and an index l, it enumerates all the ways of accommodating the elements in the set and returns the array corresponding to the position given by l.

Note that $|l|$, $|Adj(X)_{\{x_i\}}|$ and all n_j's are of the order of $\log|A|$. Therefore $U(q_w q_1 p_1 t_1 ... p_j t_j l) = X$ and

$$\begin{aligned}
K(X) &\leq |q_w p_1 t_1 ... p_j t_j l| \\
&\leq |q_w| + \sum_1^k (|q_j| + |p_j|) + |l| \\
&\leq BDM(X, \{x_i\}) + \epsilon + |q_w| \\
&+ (\log|A| + c)|Adj(X)_{\{x_i\}}| + O(\log|A|) \\
&\leq BDM(X, \{x_i\}) + O(\log^2|A|) + \epsilon,
\end{aligned}$$

which gives us the inequality.

Now, let q_X be the smallest program that generates X. For the second inequality we can describe a program $q_{\{x_i\}}$ which, given a description of X and the index j, constructs the set $Adj(X)_{\{x_i\}}$ and returns r_j, i.e. $U(q_{\{x_i\}} q_X j) = r_j$. Note that each $|j|$ is of the order of $\log|Adj(X)_{\{x_i\}}|$. Therefore, for each j we have $K(r_j) + \epsilon_j = |p_j| \leq |q_{\{x_i\}}| + |q_X| + O(\log|Adj(X)_{\{x_i\}}|)$ and $K(r_j) + \epsilon_j + \log(n_i) \leq |q_{\{x_i\}}| + |q_X| + O(\log|Adj(X)_{\{x_i\}}|) + \log(n_i)$. Finally, by adding all the terms over the j's we find the second inequality:

$$\begin{aligned}
&BDM(X, \{x_i\}) + \epsilon \leq |Adj(X)_{\{x_i\}}|(|q_X| + |q_{\{x_i\}}| \\
&+ \log(n_j) + O(\log|Adj(X)_{\{x_i\}}|)) \leq |Adj(X)_{\{x_i\}}|K(X) \\
&+ O(|Adj(X)_{\{x_i\}}| \log|Adj(X)_{\{x_i\}}|).
\end{aligned}$$

□

Corollary 7.8. *If the partition defined by $\{x_i\}$ is small, that is, if $|Adj(X)_{\{x_i\}}|$ is close to 1, then $BDM(X, \{x_i\}) \approx K(X)$.*

Proof. Given the inequalities presented in proposition 7.7, we have it that

$$K(X) - O(\log^2 |A|) - \epsilon \leq BDM(M, \{x_i\})$$

and

$$BDM(M, \{x_i\}) \leq$$
$$|Adj(X)_{\{x_i\}}|K(X) + O(|Adj(X)_{\{x_i\}}| \log |Adj(X)_{\{x_i\}}|) + \epsilon$$

which at the limit leads to $K(X)-\epsilon \leq BDM(X) \leq K(X)-\epsilon$ and $BDM(X) = K(X) - \epsilon$. From [68], we can say that the error rate ϵ is small, and that by the invariance theorem it will converge towards a constant value. □

7.2.3 Dealing with Object Boundaries

Because partitioning an object—a string, array or tensor—leads to boundary leftovers not multiple of the partition length, the only two options to take into consideration such boundaries in the estimation of the algorithmic complexity of the entire object is to either estimate the complexity of the leftovers or to define a sliding window allowing overlapping in order to include the leftovers in some of the block partitions. The former implies mixing object dimensions that may be incompatible (e.g. CTM complexity based on 1-dimensional TMs versus CTM based on higher dimensional TMs). Here we explore these strategies to deal with the object boundaries. Here we introduce a strategy for partition minimization and *base object* size maximization that we will illustrate for 2-dimensionality. The strategies are intended to overcome under- or over-fitting complexity estimations that are due to conventions, not just technical limitations (due to, e.g., incomputability and intractability).

In Section 7.2.2, we have shown that using smaller partitions for BDM yields more accurate approximations to the algorithmic complexity K. However, the computational costs for calculating CTM are high. We have compiled an exhaustive database for square matrices of size up to 4×4. Therefore it is in our best interest to find a method to minimize the partition of a given matrix into squares of size up to $d \times d = l$ for a given l.

Recursive BDM

The strategy consists in taking the biggest base matrix multiple of $d \times d$ on one corner and dividing it into adjacent square submatrices of the given size. Then we group the remaining cells into 2 submatrices and apply the same procedure, but now for $(d-1) \times (d-1)$. We continue dividing into submatrices of size 1×1.

Let X be a matrix of size $m \times n$ with $m, n \geq d$. Let's denote by quad $= \{UL, LL, DR, LR\}$ the set of quadrants on a matrix and by quad^d the set of vectors of quadrants of dimension l. We define a function $part(X, d, q_i)$, where $\langle q_1, \ldots, q_d \rangle \in$ quad^d, as follows:

$$\begin{aligned} part(X, l, q_i) = & max(X, d, q_i) \\ & \cup\, part(resL(X, d, q_i), d-1, q_{i+1}) \\ & \cup\, part(resR(X, d, q_i), d-1, q_{i+1}) \\ & \cup\, part(resLR(X, d, q_i), d-1, q_{i+1}), \end{aligned}$$

where $max(X, d, q_i)$ is the largest set of adjacent submatrices of size $d \times d$ that can be clustered in the corner corresponding to the quadrant q_i, $resR(X, d-1, q_i)$ is the submatrix composed of all the adjacent rightmost cells that could not fit on $max(X, d, q_i)$ and are not part of the leftmost cells, $resL(X, d-1, q_i)$ is an analogue for the leftmost cells and $resLR(X, d-1, q_i)$ is the submatrix composed of the cells belonging to the rightmost and leftmost cells. We call the last three submatrices *residual matrices.*

By symmetry, the number of matrices generated by the function is invariant with respect to any vector of quadrants $\langle q_1, \ldots, q_d \rangle$. However, the final BDM value can (and will) vary according to the partition chosen. Nevertheless, with this strategy we can evaluate all the possible BDM values for a given partition size and choose the partition that yields the minimum value, the maximum value, or compute the average for all possible partitions.

The partition strategy described can easily be generalized and applied to strings (1 dimension) and tensors (objects of n-dimensions).

Periodic Boundary Conditions

One way to avoid having remaining matrices (from strings to tensors) of different sizes is to embed a matrix in a topological torus (see Fig. 7.3 bottom) such that no more object borders are found. Then let X be a square matrix of arbitrary size m. We screen the matrix X for all possible combinations to minimize the number of partitions maximizing block size. We then take the combination of smallest BDM for fixed *base matrix* size d and we repeat for $d - 1$ until we have added all the components of the decomposed X. This procedure, will, however, overestimate the complexity values of all objects (in unequal fashion along the complexity spectra) but will remain bounded, as we will show in Section 7.2.4.

Without loss of generality the strategy can be applied to strings (1 dimension) and tensors (any larger number of dimensions, e.g. greater than 2), the former embedded in a cylinder while tensors can be embedded in n-dimensional tori (see Fig. 7.3).

7.2.4 Error Estimation

One can estimate the error in different calculations of BDM, regardless of the error estimations of CTM (quantified in [29, 68]), in order to calculate their departure and deviation both from granular entropy and algorithmic complexity, for which we know lower and upper bounds. For example, a maximum upper bound for binary strings is the length of the strings themselves. This is because no string can have an algorithmic complexity greater than its length, simply because the shortest computer program (in bits) to produce the string may be the string itself.

In the calculation of BDM, when an object's size is not a multiple of the base object of size d, boundaries of size $< d$ will be produced, and there are various ways of dealing with them to arrive at a more accurate calculation of an object that is not a multiple of the base. First we will estimate the error introduced by ignoring the boundaries or dealing with them in various ways, and then we will offer alternatives to take into consideration in the final estimation of their complexity.

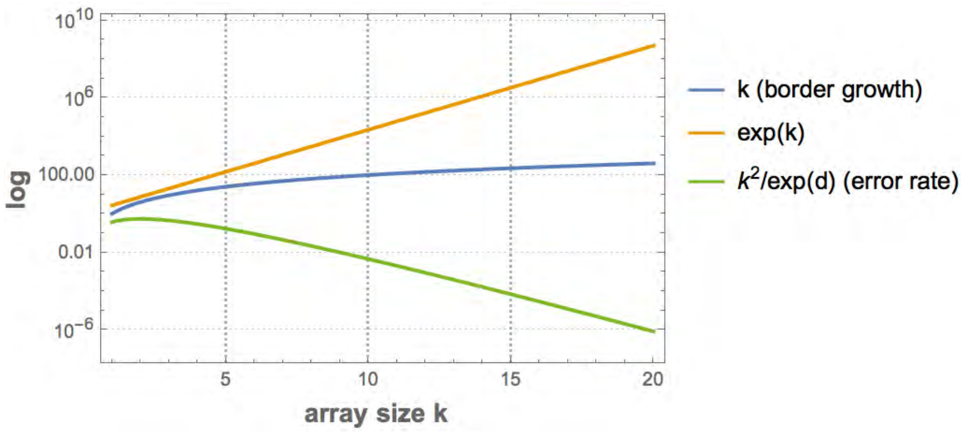

Figure 7.2: Error rate for 2-dimensional arrays. With no loss of generalization, the error rate for n-dimensional tensors $\lim_{d\to\infty}\frac{k^n}{n^k}=0$ is convergent and thus negligible, even for the discontinuities disregarded in this plot which are introduced by some BDM versions, such as non-overlapping blocks and discontinuities related to trimming the boundary condition.

If a matrix X of size $k \times j$ is not a multiple of the base matrix of size $d \times d$, it can be divided into a set of decomposed blocks of size $d \times d$, and R, L, T and B residual matrices on the right, left, top and bottom boundaries of M, all of smaller size than d.

Then boundaries R, L, T and B can be dealt with in the following way:

- Trimming boundary condition: R, L, T and B are ignored, then $BDM(X) = BDM(X, R, L, T, B)$, with the undesired effect of general underestimation for objects not multiples of d. The error introduced (see Fig. 7.2.4) is bounded between 0 (for matrices divisible by d) and $k^2/exp(k)$, where k is the size of X. The error is thus convergent ($exp(k)$ grows much faster than k^2) and can therefore be corrected, and is negligible as a function of array size as shown in Fig. 7.2.4.

- Cyclic boundary condition (Fig. 7.3 bottom): The matrix is mapped onto the surface of a torus such that there are no more boundaries and the application of the overlapping BDM version takes into consideration every part of the object. This will produce an over-estimation of the complexity of the object but will for the most part respect the ranking order of estimations if the same overlapping values are used with maximum overestimation $d - 1 \times \max\{CTM(b)|b \in X\}$, where $K(b)$ is the maximum CTM value among all base matrices b in X after thedecomposition of X.

- Full overlapping recursive decomposition: X is decomposed into $(d-1)^2$ base matrices of size $d \times d$ by traversing X with a sliding square block of size d. This will produce a polynomial overestimation in the size of the object of up to $(d-1)^2$, but if consistently applied it will for the most part preserve ranking.

- Adding low complexity rows and columns (we call this 'add col'): If a matrix of interest is not multiple the base matrices, we add rows and columns until completion to the next multiple of the base matrix, then we correct the final result by substracting the borders that were artificially added.

The BDM error rate (see 7.3 top) is the discrepancy of the sum of the complexity of the missed borders, which is an additive value of, at most, polynomial growth. The error is not even for a different complexity. For a tensor of d dimensions, with all 1s as entries, the error is bounded by $\log(k^d)$ for objects with low algorithmic randomness and by $\frac{k^d}{d^k}$ for objects with high algorithmic randomness.

Ultimately there is no optimal strategy for making the error disappear, but in some cases the error can be better estimated and corrected 7.2.4 and all cases are convergent, hence asymptotically negligible, and in all cases complexity ranking is preserved and under- and over-estimations bounded.

7.2.5 BDM Convergence towards Shannon entropy

Let $\{x_i\}$ be a partition of X defined as in the previous sections for a fixed d. Then the Shannon entropy of X for the partition $\{x_i\}$ is given by:

$$H_{\{x_i\}}(X) = - \sum_{(r_j,n_j) \in Adj(X)_{\{x_i\}}} \frac{n_j}{|\{x_i\}|} \log(\frac{n_j}{|\{x_i\}|}), \tag{7.6}$$

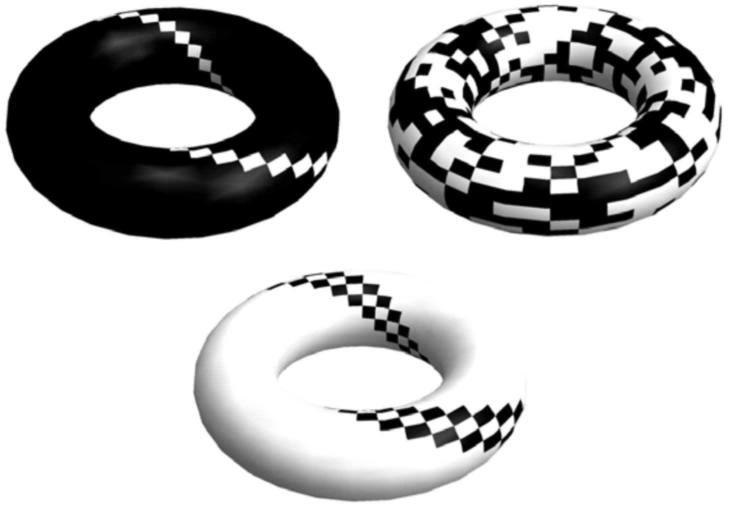
Figure 7.3: One way to deal with the decomposition of n-dimensional tensors is to embed them in an n-dimensional torus($n = 2$ in the case of the one depicted here), making the borders cyclic or periodic by joining the borders of the object. Depicted here are three examples of graph canonical adjacency matrices embedded in a 2-dimensional torus that preserves the object complexity on the surface, a complete graph, a cycle graph and an Erdös-Rényi graph with edge density 0.5, all of size 20 nodes and free of self-loops. Avoiding borders has the desired effect of producing no residual matrices after the block decomposition with overlapping.

where $P(r_j) = \frac{n_j}{|\{x_i\}|}$ and the array r_j is taken as a symbol itself. The following proposition establishes the asymptotic relationship between $H_{\{x_i\}}$ and BDM.

Proposition 7.9. *Let M be a 2-dimensional matrix and $\{x_i\}$ a partition strategy with elements of maximum size $d \times d$. Then:*

$$|BDM_{\{x_i\}}(X) - H_{\{x_i\}}(X)| \leq O(\log(|\{x_i\}|))$$

Proof. First we note that $\sum n_j = |\{x_i\}|$ and, given that the set of matrices of size $d \times d$ is finite and so is the maximum value for $CTM(r_j)$, there exists a constant c_d such that $|Adj(X)_{\{x_i\}}|CTM(r_j) < c_d$. Therefore:

$$\begin{aligned}
&BDM_{\{x_i\}}(X) - H_{\{x_i\}}(X) \\
&= \sum (CTM(r_j) + \log(n_j) + \frac{n_j}{|\{x_i\}|}\log(\frac{n_j}{|\{x_i\}|})) \\
&\leq c_d + \sum(\log(n_j) + \frac{n_j}{|\{x_i\}|}\log(\frac{n_j}{|\{x_i\}|})) \\
&= c_d + \sum(\log(n_j) - \frac{n_j}{|\{x_i\}|}\log(\frac{|\{x_i\}|}{n_j})) \\
&= c_d + \frac{1}{|\{x_i\}|}\sum(|\{x_i\}|\log(n_j) - n_j\log(\frac{|\{x_i\}|}{n_j})) \\
&= c_d + \frac{1}{|\{x_i\}|}\sum\log(\frac{n_j^{|\{x_i\}|+n_j}}{|\{x_i\}|^{n_j}})
\end{aligned}$$

Now, let's recall that the sum of n_j's is bounded by $|\{x_i\}|$. Therefore exists c_d' such that

$$\begin{aligned}
\frac{1}{|\{x_i\}|}\sum\log(\frac{n_j^{|\{x_i\}|+n_j}}{|\{x_i\}|^{n_j}}) &\leq \frac{c_d}{|\{x_i\}|}\log(\frac{|\{x_i\}|^{|\{x_i\}|+c_d'|\{x_i\}|}}{|\{x_i\}|^{c_d'|\{x_i\}|}}) \\
&= \frac{c_d}{|\{x_i\}|}\log(|\{x_i\}|^{|\{x_i\}|}) \\
&= c_d\log(|\{x_i\}|).
\end{aligned}$$

□

Now, is important to note that the previous proof sets the limit in terms of the constant c_d, which minimum value is defined in terms of matrices for which the CTM value has been computed. The smaller this number is, the tighter is the bound set by 7.9. Therefore, in the worst case, this is when CTM has been computed for a comparatively small number of matrices, or the larger base matrix have small algorithmic complexity, the behavior of BDM is similar to entropy. In the best case, when CTM is updated by any means, BDM approximates algorithmic complexity (corollary 7.8).

Furthermore, we can think on $c_d\log(|\{x_i\}|)$ as a measure of the deficit in information incurred by both, entropy and BDM, in terms of each other. Entropy is missing the number of base objects needed in order to get an approximation of the compression length of M, while BDM is missing the position of each base symbol. And giving more information to both measures wont necessarily yield a better approximation to K.

7.2.6 BDM versus Entropy (Rate) and Compression

Let us address the task of quantifying how many strings with maximum entropy rate are actually algorithmically compressible, i.e., have low algorithmic complexity. That is, how many strings are actually algorithmically (as opposed to simply statistically) compressible but are not compressed by lossless compression algorithms, which are statistical (entropy rate) estimators. We know that most strings have both maximal entropy (most strings look equally statistically disordered, a fact that constitutes the foundation of thermodynamics) and maximal algorithmic complexity (according to a pigeonhole argument, most binary strings cannot be matched to shorter computer programs as these are also binary strings). But the gap between those with maximal entropy and low algorithmic randomness diverges and is infinite at the limit (for an unbounded string sequence). That is, there is an infinite number of sequences that have maximal entropy but low algorithmic complexity.

The promise of BDM is that, unlike compression, it does identify some cases of strings with maximum entropy that actually have low algorithmic complexity. Fig. 7.4 shows that indeed BDM assigns lower complexity to more strings than entropy, as expected. Unlike entropy, and implementations of lossless compression algorithms, BDM recognizes some strings that have no statistical regularities but have algorithmic content that makes them algorithmically compressible.

7.3 Approximating $m(s)$

There are thus three methods available today for approximating K:

- CTM deals with all 2 bit strings of length 1-12 (and for some 20-30

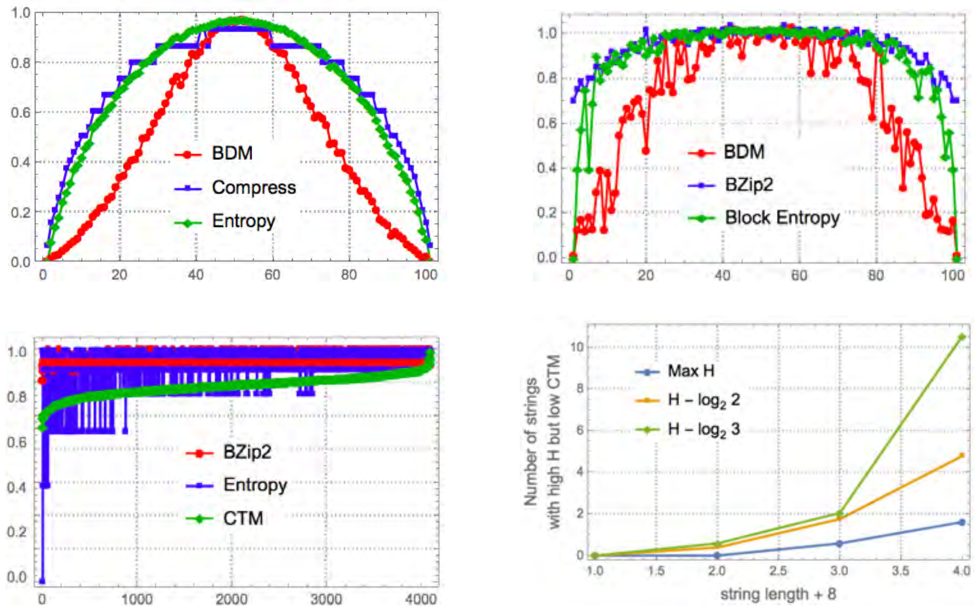

Figure 7.4: Top left: Comparison between values of entropy, compression (Mathematica's `Compress[]`) and BDM over a sample of 100 strings of length 10 000 generated from a binary random variable following a Bernoulli distribution and normalized by maximal complexity values. Entropy just follows the Bernoulli distribution and, unlike compression that follows entropy, BDM values produce clear convex-shaped gaps on each side assigning lower complexity to some strings compared to both entropy and compression. Top right: The results confirmed using another popular lossless compression algorithm BZip2. Bottom right: When strings are sorted by CTM, one notices that BZip2 collapses most strings to minimal compressibility. Over all $2^{12} = 4096$ possible binary strings of length 12, entropy only produces 6 different entropy values, but CTM is much more fine-grained, and this is extended to the longer strings by BDM.Similar results were obtained when using a third lossless compression algorithm, LZMA.

bits).

- BDM deals with 12 bits to hundreds of bits (with a cumulative error that grows by the length of the strings–if not applied in conjunction with CTM). The worst case occurs when substrings share information content with other decomposed substrings and BDM just keeps adding their K values individually.

- Lossless compression deals with no less than 100 bits and is unstable up to about 1K bits.

While CTM cannot produce estimations of longer bitstrings, estimating the algorithmic complexity of even bitstrings can be key to many problems.

Because BDM locally estimates algorithmic complexity via algorithmic probability based upon CTM, it is slightly more independent of object description than computable measures such as Shannon entropy, though in the 'worst case' it behaves like Shannon entropy. We have also shown that the various flavours of BDM are extremely robust, both by calculating theoretical errors on tensors and by numerical investigation, **establishing that any BDM version is fit for use in most cases**. Hence the most basic and efficient one can be used without much concern as to the possible alternative methods that could have been used in its calculation, as we have exhaustively and systematically tested most, if not all, of them.

Given the fitness showed by BDM, we can approximate the universal distribution $m(s)$ by

$$m(s) \approx \frac{1}{2^{BDM(s)}},$$

where $BDM(s)$ is one of the variants of BMD applied to the object s.

In the next chapter, I will proceed to define an stochastic dynamical system using this approximation in what correspond to the first numerical experimental in algorithmic probability driven evolution.

Chapter 8

A First Experiment in Algorithmic Evolution

Whenever the universe can be considered computable —or algorithmic— and the nature of its observed randomness are deep metaphysical questions. In this chapter I will show what, to the best of my knowledge, are the firsts numerical experiments that compare the algorithmic origin of random mutations with the alternative hypothesis. These experiments, along with derived theoretical analysis, show that an algorithmic origin for the randomness occurring in the natural world has the potential to better explain diverse phenomenon observed in natural evolution.

Among the phenomena better explained by algorithmic random mutations are the faster than expected emergence of deep structures that are repeated over subsequences generations, modularity, genetic memory, and periods of accelerated grow in diversity and accelerated extinction.

The content of the present chaper overlaps with the article the article '***Algorithmically probable mutations reproduce aspects of evolution such as convergence rate, genetic memory, modularity, diversity explosions, and mass extinction***' [41], for which I was first author.

8.1 Why Algorithmic Evolution?

Central to modern synthesis and general evolutionary theory is the understanding that evolution is gradual and is explained by small genetic changes in populations over time [38]. Genetic variation in populations can arise by chance through mutation, with these small changes leading to major evolutionary changes over time.

Of interest in connection to the possible links between the theory of biological evolution and the theory of information is the place and role of randomness in the process that provides the variety necessary to allow organisms to change and adapt over time.

It has been suggested [23, 76, 79, 82] that the deeply informational and computational nature of biological organisms makes them amenable to being studied or considered as computer programs following (algorithmic) random walks in software space, that is, the space of all possible—and valid—computer programs.

The following model will allow me to numerically test this hypothesis and explore the possible consequences of its validation vis-a-vis our understanding of the biological aspects of life and natural evolution by natural selection, as well as for applications to optimization problems in areas such as evolutionary programming.

8.2 Methodology

8.2.1 Theoretical Considerations

As sated in chapter 6, Chaitin's evolutionary model [21, 23, 22], a successful mutation is defined as a computable function μ, chosen according to the probabilities stated by the Universal Distribution (definition 6.1), that changes the current state of the system (as an input of the function) to a better approximation of the constant Ω (definition 6.3). In order to be able to simulate this system we would need to compute the Universal Distribution and the fitness function. However, both the Universal Distribution and the fitness function of the system require the solution of the Halting Problem [74], which is incomputable. Nevertheless, as with Ω itself, this solution can be approximated [14, 86]. In this chapter I will show a model that, to the best of my knowledge, is the first computable approximation to Chaitin's proposal.

For this approximation I have made four important initial concessions: one with respect to the *real* computing time of the system, and three with respect to Chaitin's model:

- I assume that building the probability distributions for each instance of the evolution takes no computational time, while in the *real computation* this is the single most resource-intensive step.
- The goal of the system is to approximate objects of bounded information content: binary matrices of a set size.
- I use BDM and Shannon's entropy as approximations for the algorithmic information complexity K.
- I am not approximating the algorithmic probability of the mutation functions, but that of their outputs.

I justify the first concession in a similar fashion as Chaitin: if we assume that the interactions and mechanics of the natural world are computable, then the probability of a decidable event occurring is given by the Universal Distribution. The third one is a necessity, as the algorithmic probability of an object is incomputable (it requires a solution for HP too). In the next section I will argue that Shannon's entropy is not as good as BDM for my purposes. Finally, note that given the universal distribution and a fixed input, the probability of a mutation is in inverse proportion to the descriptive complexity of its output, up to a constant error. In other words, it is highly probable that a mutation may reduce the information content of the input but improbable that it may increase the information content. Therefore, the last concession yields an adequate approximation, since a low information mutation can reduce the descriptive complexity of the input but not increase it in a meaningful way.

8.2.2 The Expectations

It is important to note that, when compared to Chaitin's metabiology model, I had to change the goal of my system therefore I also had to change the expectations I had for its behavior.

Chaitin's evolution model (chapter 6) is faster than *regular* random models despite targeting a highly random object (Ω, definition 6.3), thanks to the fact that positive mutations have low algorithmic information complexity and hence a (relatively) high probability of being stochastically chosen under the Universal Distribution. The universally low algorithmic complexity of these positive mutations relies on the fact that, when assuming an oracle for HP, we are also implying a constant algorithmic complexity for its evaluation function and target, since we can write a program that verifies if a change on a given approximation of Ω is a positive one without needing a codification of Ω itself.

In contrast, my model is expected to be sensitive with respect to the algorithmic complexity of the target matrix, obtaining high speed-up for structured target matrices that decreases as the algorithmic complexity of the target grows. However, this change of behavior remains congruent with the main argument of metabiologym and our assertion that, contrary to *regular random* mutations, algorithmic probability driven evolution tends to produce structured novelty at a faster rate, which we hope to prove in the upcoming set of experiments.

In summary, I expected that when using an approximation to the Universal Distribution:

- Convergence will be reached in significantly fewer total mutations than when using the uniform distribution for structured target matrices.

- The stated difference will decrease in relation to the algorithmic complexity of the target matrix.

The Unsuitability of Shannon's Entropy

As shown in [80, 86] (chapter 7), when compared to BDM we can think of Shannon's entropy alone as a less accurate approximation to the algorithmic complexity of an object (if its underlying probability distribution is not updated by a method equivalent to BDM, as it would not be by the typical uninformed observer). Therefore I expected the entropy-induced speed-up to be consistently outperformed by BDM when the target matrix inclines away from algorithmic randomness and has thus some structure. Furthermore, as random matrices are expected to have a balanced number of 0's and 1's, I anticipated the performance of single bit entropy to be nearly identical to the uniform distribution on unstructured (random) matrices. For block entropy [66, 65], that is, the entropy computed over submatrices rather than single bits, the probability of having repeated blocks is in inverse proportion to their size, while blocks of smaller sizes approximate single bit entropy, again yielding similar results to the uniform distribution. The results support my assumptions and claims.

8.2.3 Evolutionary Model

Broadly speaking, my evolutionary model is a tuple $\langle S, \mathbb{S}, M_0, f, t, \alpha \rangle$, where:

- $\mathbb{S}$ is the *state space*,
- M_0, with $M_0 \in \mathbb{S}$, is the *initial state* of the system,
- $f : \mathbb{S} \mapsto \mathbb{R}^+$ is a function, called the *fitness or aptitude function*, which goes from the state space to the positive real numbers,
- t is a positive integer called the *extinction threshold*,
- α is a real number called the convergence parameter, and
- $S : \mathbb{S} \mapsto \mathbb{S} \times (\mathbb{Z}^+ \cup \{\perp, \top\})$ is a non-deterministic evolution dynamic such that if $S(M, f, t) = (M', t')$ then $f(M') < f(M)$ and $t' \leq t$, where t' is the number of *steps or mutations* it took S to produce M', $S(M, f, t) = (\perp, t')$ if it was unable find M' with a *better fitness* in the given time, and $S(M, f, t) = (\top, t')$ if it finds M' such that $f(M') \leq \alpha$.

Specifically, the function S receives an individual M and returns an *evolved individual* M', in the time specified by t, that improves upon the value of the fitness function f and the time it took to do so, $\perp$ if it was unable to do so and $\top$ if it reached the convergence value.

A *successful evolution* is the sequence

$$M_0, (M_1, t_1) = S(M_0, f, t), ..., (\top, t_n))$$

and $\sum t_i$ is the total evolution time. We say that the evolution failed, or that we got an *extinction*, if instead we finish the process by $(\perp, t_n)$, with $\sum t_i$ being the extinction time. The evolution is undetermined otherwise. Finally, we will call each element (M_i, t_i) an *instance* of the evolution.

8.2.4 Experimental Setup: A Max One Problem Instance

For this experiment, our phase state is the set of all binary matrices of sizes $n \times n$, our fitness function is defined as the Hamming distance

$$f(M) = H(M_t, M),$$

where M_t is the *target matrix*, and our convergence parameter is $\alpha = 0$. In other words, the evolution converges when we produce the target matrix, guided only by the Hamming distance to it, which is defined as the number of different bits between the *input matrix* and the target matrix.

The stated setup was chosen since it allowed me to easily define and control the descriptive complexity of the fitness function by controlling the target matrix and, therefore also control the complexity of the evolutionary system itself. It is important to note that our setup can be seen as a generalization of the *Max One problem* [64], where the initial state is a binary "target gene" and the target matrix is the "target gene"; when we obtain a Hamming distance of 0 we have obtained the gene equality.

8.2.5 Evolution Dynamics

The main goal of this experiment was to contrast the speed of the evolution when choosing between two approaches to determining the probability of mutations:

- When the probability of a given set of mutations has a *uniform distribution.* That is, all possible mutations have the same probability of occurrence, even if under certain constraints.
- When the probability of a given mutation occurring is given by an approximation to the universal distribution. As the UD is non-computable, we will approximate it by approximating the algorithmic complexity K by means of the Block Decomposition Method (BDM, with no overlapping, chapter 7).
- I will also investigate the results by running a number of experiments using Shannon Entropy instead of BDM to approximate K.

Each *evolution instance* was computed by iterating over the same dynamic. I started by defining the set of possible mutations as those that are within a fixed n number of bits from the input matrix. In other words, for a given input matrix M, the set of possible mutations in a single instance is defined as the set

$$\mathbb{M}(M) = \{M' | H(M', M) \leq n\}.$$

Then, for each matrix in $\mathbb{M}$, I computed the probability $P(M')$ is defined as:

- $P(M) = \frac{1}{|\mathbb{M}|}$ in the case of the Uniform Distribution.
- $P(M) = \frac{\beta}{2^{BDM(M)}}$ for the BDM Distribution and
- $P(M) = \frac{\beta'}{h(M)}$ or $P(M) = \frac{\beta''}{2^{h(M)}}$ for Shannon entropy (for an uninformed observer with no access to the possible deterministic or stochastic nature of the source),

where β, β' and β'' are normalization factors such that the sum of the respective probabilities are 1.

For implementation purposes, I used a minor variation to the entropy probability distribution to be used and compared to BDM. The probability distributions for the set of possible mutations using entropy were built using two heuristics: Let M' be a possible mutation of M, then the probability of obtaining M' as a mutation is defined as either, $\frac{\beta'}{h(M')+\epsilon}$ or $\frac{\beta''}{2^{h(M')}}$. The first definition assigns a linearly higher probability to mutations with lower entropy. The second definition is consistent with our use of BDM in the rest of the experiments. The constant ϵ is an arbitrary small value that was included to avoid undefined (infinite) probabilities. For the presented experiments ϵ was set at 1^{-10}.

Once the probability distribution was computed, I set the number of steps as 0 and then, using a (pseudo)random number generator (RNG), we proceed to stochastically draw a matrix from the sated probability distributions and evaluate its fitness with the function f, adding 1 to the number of steps. If the resultant matrix does not show an improvement in fitness, I draw another matrix and add another 1 to the number of steps, not stopping the process until we obtain a matrix with better fitness or reach the extinction threshold. I can choose to either replace the drawn matrix or leave it out of the pool for the next iterations. A visualization of the stated work flow for a 2×2 matrix is shown in Figure 8.1.

To get a complete evolution sequence, I iterated the stated process until either convergence or extinction is reached. As stated before, I could choose to not replace an evaluated matrix from the set of possible mutations in each instance, but I chose to not keep track of evaluated matrices after an instance was complete. This was done in order to keep open the possibility of dynamic fitness functions in future experiments.

In this case, the *evolution time* is defined as the sum of the number of *steps* (or draws) it took the initial matrix to reach equality with the target matrix. When computing the evolution dynamics by one of the different probability distribution schemes we will denote it by *uniform strategy*, *BDM strategy* or *h strategy*, respectively. That is, the uniform distribution, the distribution for the algorithmic probability estimation by BDM, and the distribution by Shannon entropy.

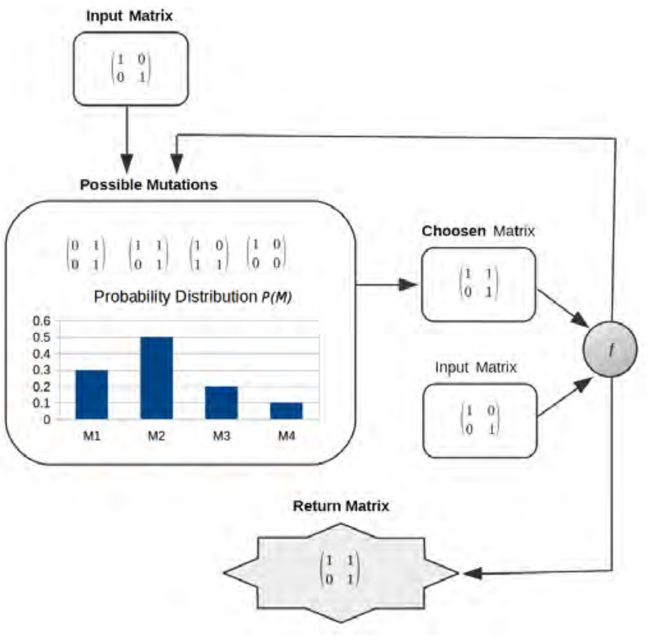

Figure 8.1: An evolution instance. The instances are repeated by updating the input matrix until convergence or extinction is reached.

8.2.6 The Speed-Up Quotient

We will measure how fast (or slow) a strategy is compared to the uniform by the speed-up quotient, which I define as:

Definition 8.1. The speed-up quotient, or simply *speed-up*, between the uniform strategy and a given strategy f is defined as

$$\delta = \frac{S_u}{S_f},$$

where S_u is the average number of steps it takes a sample (a set of initial state matrices) to reach convergence under the uniform strategy and S_f is the average number of steps it takes under the f strategy.

Table 8.1: Results obtained for the 'Random Graphs'

Strategy	Shifts	Average	SE
Uniform	1	214.74	3.55
h_b	1	214.74	3.55
h	1	215.53	3.43
h_b^2	1	214.74	3.55
h^2	1	213.28	3.33
Uniform	2	1867.10	78.94
h_b	2	1904.52	79.88
h	2	2036.13	83.38
h_b^2	2	1882.46	78.63
h^2	2	1776.25	81.93

8.3 Results

8.3.1 Cases of Negative Speed-up

In order to better explain the choices I have made to our experimental setup, first I will present a series of cases where I obtained no speed-up or slow-down. Although these cases were expected, they shed important light on the behavior of the system.

Entropy vs. Uniform on Random Matrices

For the following experiments, I generated 200 random matrices separated into two sets: *initial matrices* and *target matrices*. After pairing them based on their generation order I evolved them using 10 strategies: the uniform distribution, *block* Shannon's entropy for blocks of size 4×4, denoted below by h_b, entropy for single bits denoted by h, and their variants where we divide by h^2 and h_b^2 respectively. The strategies were repeated for 1-bit and 2-bit shifts (mutations).

The results obtained are summarized in the table 8.1, which lays out the strategy used for each experiment, the number of shifts/mutations allowed, the average number of steps it took to reach convergence, as well as the standard error of the sample mean. As we can see, the differences in the number of steps required to reach convergence are not significant, validating our assertion that, for random matrices, entropy evolution is not much different than the uniform evolution.

Because the algorithmic complexity of a network makes sense only in its unlabeled version in general, and in most of the cases. In [85, 81, 86] it was shown, both theoretically and numerically, that approximations of algorithmic complexity of adjacency matrices of labeled graphs are a good approximation (up to a logarithmic term or the numerical precision of the algorithm) of the algorithmic complexity of the unlabeled graphs. This means that we can consider any adjacency matrix of a network a good representation of the network disregarding graph isomorphisms.

Entropy vs. Uniform on a Highly Structured Matrix

For this set of experiments, I took the same set of 100 8×8 initial matrices and evolved them into a highly structured matrix, which is the adjacency matrix of the star with 8 nodes. For this matrix, I expected entropy to be unable to capture its structure, and the results obtained accorded with my expectations. The results are shown in table 8.2.

As we can see from the results, entropy was unable to show a statistically significant speed-up compared to the uniform distribution. Over the next sections I will show that I have obtained a statistically significant speed-up by using the BDM approximation to algorithmic probability distributions, which is expected because *BDM manages to better capture the algorithmic structures of a matrix rather than just the distribution of the bits which entropy measures.* Based on the previous experiments, I conclude that entropy is thus not a good approximation for K, and we will omit its use in the rest of the article.

Table 8.2: Results obtained for the 'Star'

Strategy	Shifts	Average	SE
Uniform	1	216.24	3.48
h_b	1	216.71	3.54
h	1	212.74	3.41
h_b^2	1	216.71	3.54
h^2	1	211.74	3.69
Uniform	2	1811.84	85.41
h_b	2	1766.69	88.18
h	2	1859.11	75.73
h_b^2	2	1764.03	84.52
h^2	2	1853.04	74.48

8.3.2 The Causes of Extinction

For the uniform distribution, the reason is simple: the number of 3-bit shifts on 8×8 matrices gives a space of possible mutations of $\binom{8\times 8}{3} = 41664$ matrices, which is much larger than the number of possible mutations present within 2-shifts and 1-shift (mutation), which are $\binom{8\times 8}{2} = 2016$ and $8 \times 8 = 64$ respectively. Therefore, as we get close to convergence, the probability of getting the right evolution, if the needed number of shifts is two or one, is about 0.04%, and removing repeated matrices does not help in a significant way to avoid extinction,since 41 664 is much larger than 2500.

Given the values discussed, I chose to set the extinction threshold at 2500 and the number of shifts at 2 for 8×8 matrices, as allowing just 64 possible mutations for each stage is a number too small for showing a significant difference in the evolutionary time between the uniform and BDM strategies, while requiring evolutionary steps of ˜41 664 for an evolutionary stage is too computationally costly. The threshold of 2500 is close to the number of possible mutations and has been shown to consume a high amount of computational resources. For 16×16 matrices, I performed 1-bit shifts only, and occasionally 2-bit shifts when computationally possible.

The BDM Strategy, Extinctions and Persistent Structures

The interesting case is the BDM strategy. As we can see clearly in Figure 8.2 for the 8×8 3-bit case, the overall number of steps needed to reach each extinction is often significantly higher than 2500 under the BDM strategy. This behavior cannot be explained by the analysis done for the uniform distribution, which predicts the sharp drop observed in the blue curve.

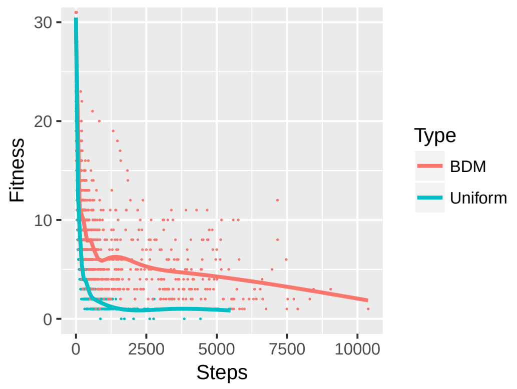

Figure 8.2: Fitness graph for random 8×8 matrices with 3-bit shifts (mutations). Evolution convergence is reached at a fitness of 0. The curves are polynomial approximations computed for visual aid purposes.

After analyzing the set of matrices drawn during failed mutations (all the matrices drawn during a single failed evolutionary stage), I found that most of these matrices have in common highly regular structures. We will call these structures *persistent structures.* Formally, regular structures can be defined as follows:

Definition 8.2. Let M be the description used for an organism or population and Γ a substructure of M in a *computable position* such that $K(M) = K(\Gamma)+K(M-\Gamma)-\epsilon$, where ϵ is a *small* number and $M-\Gamma$ is the codification M without the contents of S. We will call Γ a *regular structure* of degree γ if the probability of choosing a mutation M' with the subsequence Γ is $1 - 2^{-(\gamma-\epsilon)}$.

Now, note that γ grows in inverse proportion to $K(\Gamma)$ and the difference in algorithmic complexity of the mutation candidates and $K(\Gamma)$: Let M

contain Γ in a computable position. Then the probability of choosing M' as an evolution of M is

$$\frac{1}{2^{K(M')}} \geq \frac{1}{2^{K(M'-\Gamma)+K(\Gamma)+O(1)}}.$$

Furthermore, if the possible mutations of M can only mutate a bounded number of bits and there exists C such that, for every other subsequence of Γ' that can replace Γ we have it that $K(\Gamma') \geq K(\Gamma) + C$, then:

$$P(M' \text{ contains } \Gamma) \geq 1 - O(2^{-C}).$$

The previous inequality is a consequence of the fact that the possible mutations are finite and only a small number of them, if any, can have a smaller algorithmic complexity than the mutations that contain Γ; otherwise we contradict the existence of C. In other words, as Γ has relatively *low complexity*, the structures that contain Γ tend to also have low algorithmic complexity, and hence a higher probability of being chosen.

Finally, as shown in the section 8.3.2, *we can expect the number of mutations with persistent structures to increase in factorial order with the number of possible mutations and in polynomial order with respect to the size of the matrices that compose the state space.*

Proposition 8.3. *As a direct consequence of the last statement, we have it that, for systems evolving as described in the section 8.2.5 under the Universal Distribution:*

- *Once a structure with low descriptive complexity is developed, it is* exponentially hard *to get rid of it.*
- *The probability of finding a mutation without the structure decreases* in factorial order *with respect to the set of possible mutations.*
- *Evolving towards random matrices is* hard (improbable).
- *Evolving from and to* unrelated *regular structures is also* hard.

Given the fourth point, we will always choose random initial matrices from now on, as the probability of drawing a mutation other than an empty matrix (of zeroes), when one is present in the set of possible mutations, is extremely low (below 9×10^{-6} for 8×8 matrices with 2 shifts).

Randomly Generated Graphs

For this set of experiments, I generated 200 random 8×8 matrices and 600 16×16 matrices, both sets separated into initial and target matrices. I then proceeded to evolve the initial matrix into the corresponding target by the following strategies: uniform and BDM within 2-bit and 3-bit shifts (mutations) for the 8×8 matrices and only 2-bit shifts for the 16×16 matrices due to computing time. The results obtained are shown in the Figure 8.3. In all cases, I did not replace drawn matrices and the extinction threshold was set at 2500.

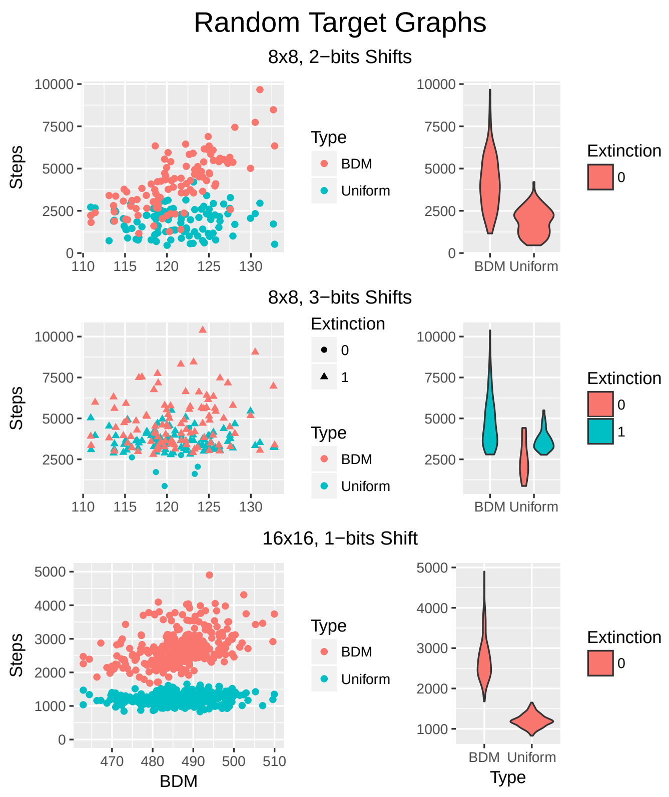

Figure 8.3: Randomly generated 8×8 and 16×16 matrices.

From the results we can see two important behaviors for the 8×8 matrices. The matrices generated are of high BDM complexity and evolving the system using the uniform strategy tends to be faster than using BDM for these highly random matrices. Secondly, although increasing the number of possible shifts by 1 seems, at a first glance, a small change in our setup, it has a big impact on our results: the number of extinctions has gone from 0 for both methods to 92 for the uniform strategy and 100 for BDM. This means that most evolutions will rise above our threshold of 2500 drafts for a single successful evolutionary step, leading to an extinction. As for the 16×16 matrices, we can see a formation of two easily separable clusters that coincide perfectly with the Uniform and BDM distributions respectively.

8.3.3 Positive Speed-Up Instances

In the previous section, we established that the BDM strategy yields a negative speed-up when targeting randomly generated matrices, which are expected to be of high algorithmic information content or *unstructured*. However, as stated in section 8.2.2, that behavior is within our expectations. In the next section I will show instances of positive speed-up, including cases where previously entropy failed to show significant speed-up or was outperformed by BDM.

Manually Built Structured Matrices

For the following set of experiments I manually built three 8×8 matrices that encode the adjacency matrices of three undirected non-random graphs with 8 nodes that are intuitively *structured*: the *complete graph*, the *star graph* and a *grid*. The matrices used are shown in Figure 8.4.

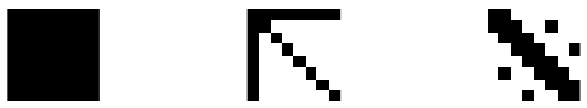

Figure 8.4: Adjacency matrices for the labelled complete, star, and grid graphs.

After evolving the same set of 100 randomly generated matrices for the three stated matrices, I can report that I found varying degrees of positive speed-up, that correspond to their respective descriptive complexities as approximated by their BDM values. The complete graph, along with the empty graph, is the graph that has the lowest approximated descriptive complexity with a BDM value of just 24.01. As expected, we get the best speed-up quotient in this case. After the complete graph, the star intuitively seems to be one of the less complex graphs we can draw. However, its BDM value (105.434) is significantly higher than the grid (83.503). Accordingly, the speed-up obtained is lower. The results are shown in the Figure 8.5.

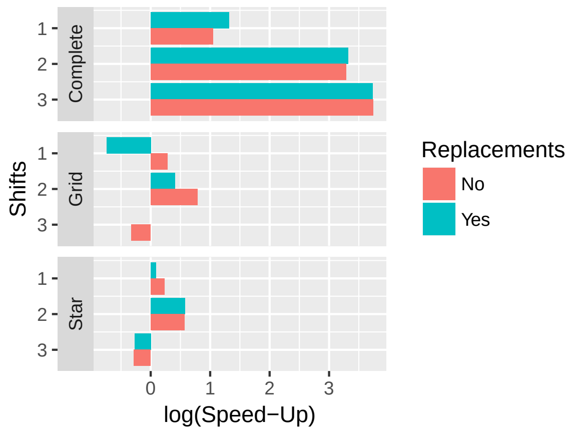

Figure 8.5: The logarithm of the speed-up obtained for the matrices in Figure 8.4.

As we can see from the Figure 8.5, a positive speed-up quotient was consistently found within 2-bit shifts without replacements. We have one instance of negative speed-up with one shift with replacements for the grid, and negative speed-up for all but the complete graph with two shifts.

However, it is important to say that almost all the instances of negative speed-up are not statistically significant, as we have a very high extinction rate of over 90%, and the difference between the averages is lower than two standard errors of the mean. The one exception is the grid at 1-bit shift, which had 45 extinctions for the BDM strategy. The complete tables are presented in the appendix of the full article [41].

8.3.4 Mutation Memory

The cause of the extinctions found in the grid are what I will call *maladaptive* persistent structures (definition 8.2), as they occur at a significantly higher rate under the BDM distribution. Also, as the results suggest, a strategy to avoid this problem is adding *memory* to the evolution. In our case, I will not replace matrices already drawn from the set of possible mutations.

I do not believe this change to be contradictory to the stated goals, since another way to see this behavior is that the universal distribution *dooms* (with very high probability) *populations with certain mutations to extinction*, and *evolution must find strategies* to eliminate these mutations fast from the population. This argument also implies that extinction is faster under the universal distribution than regular random evolution when a persistent maladaptive mutation is present, which can be seen as a form of mutation memory. This requirement has the potential to explain evolutionary phenomena such as the Cambrian explosion, as well as mass extinctions: once a positively structured mutation is developed, further algorithmic mutations will keep it (with a high probability), and the same applies to negatively structured mutations. The same mechanic can also explain the recurring structures found in the natural world. Degradation of a structure is still possible, but will be relatively slow. In other words, *evolution will remember positive and negative mutations (up to a point) when they are structured.*

From now on, I will assume that our system has memory (no replacement) and that mutations are not replaced when drawn from the distribution.

8.3.5 The Speed-Up Distribution

Having explored various cases, and found several conditions where negative and positive speed-up are present, the aim of the following experiment was to offer a broader view of the distribution of speed-up instances as functions of their algorithmic complexity.

For the 8×8 case, I generated 28 matrices by starting with the undirected complete graph with 8 nodes, represented by its adjacency matrix, and then

I removed one edge at a time until the empty graph (the diagonal matrix) was left, obtaining our *target matrix set*. It is important to note that the resultant matrices are always symmetrical. The process was repeated for the 16×16 matrices, obtaining a total of 120 16×16 target matrices.

For each target matrix in the first *target matrix* set, I generated 50 random initial matrices and evolved the population until convergence was reached using the two stated strategies: uniform and BDM, both without replacements. I saved the number of steps it took for each of the 2800 evolutions to reach convergence and computed the average speed-up quotient for each target matrix. The stated process was repeated for the second target matrix set, but by generating 20 random matrices for each of the 120 target matrices to conserve computational resources. The experiment was repeated for shifts of 1 and 2 bits and the extinction thresholds used were 2500 for 8×8 and 10 000 for 16×16 matrices.

As we can see from the results in Figure 8.6, the average number of steps required to *reach convergence* is significantly lower when using the BDM distribution for matrices with low algorithmic complexity, but the difference drops along with the complexity of the matrices but never crosses the *extinction threshold*. This suggests that symmetry over the diagonal is enough to guarantee a degree of structure that can be captured by BDM. It is important to report that I found no extinction case for the 8×8 matrices, 13 in the 16×16 matrices with 1-bit shifts, all for the BDM distribution, and 1794 with 2-bit shifts, mostly for the uniform distribution.

This last experiment was computationally very expensive. Computing the data required for the 16×16, 2-bit shifts sequence took 12 days, 6 hours and 22 minutes on a single core of an i5-4570 PC with 8GB of RAM. Repeating this experiment for 3-bit shifts is infeasible with our current setup, as it would take us roughly two months shy of 3 years.

Now, by combining the data obtained for the previous sequence and the random matrices used in section 8.3.2, we can approximate the positive speed-up distribution. Given the nature of the data, this approximation (Figure 8.7) is given as two curves, each representing the expected evolution time from a random initial matrix as a function of the algorithmic information complexity of the target matrix for both strategies, uniform and BDM respectively. The

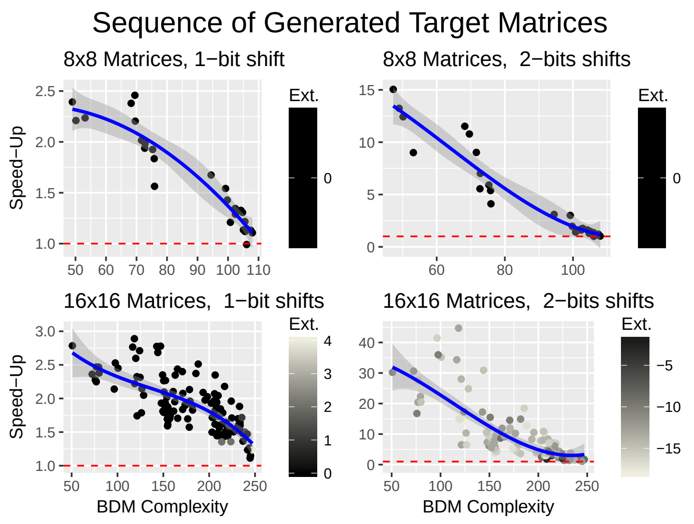

Figure 8.6: The *Speed-Up quotient* is defined as $\delta = \frac{S_u}{S_{BDM}}$, where S_u is the average number of steps it took to reach convergence under the uniform strategy and S_{BDM} for BDM, and *'Ext.'* is the difference $E_u - E_{BDM}$ where each factor is the number of extinctions obtained for the universal and BDM distribution, respectively. In the case of an extinction, the sample was not used to compute the average number of steps. The red (dashed) line designates the *Speed-Up threshold at* $y = 1$: above this line we have positive speed-up and below it we have negative speed-up. The blue (continuous) line represents a cubic fit by regression over the data points.

positive speed-up instances are those where the the BDM curve is below the uniform curve.

The first result we get from Figure 8.7 is a confirmation of an expected behavior: unlike the uniform strategy, the BDM strategy is highly sensitive to the algorithmic information content of the target matrix. In other words, *it makes no difference for a uniform probability mutation space whether the solution is structured or not, while an algorithmic probability driven mutation will naturally converge faster to structured solutions.*

The results obtained expand upon the theoretical development presented in section 8.3.2. As the set of possible mutations grows, so do the instances of persistent structures and the slow-down itself. This behavior is evident given

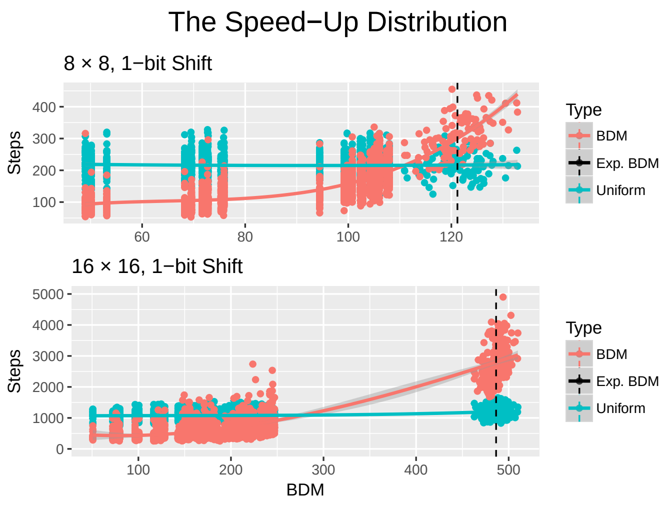

Figure 8.7: The positive speed-up instances are those where the coral curve, computed as a cubic linear regression to all the evolution times for the BDM strategy, are below the teal line, which is a cubic approximation to the evolution times for the uniform strategy. The black line is the *expected* BDM value for a randomly chosen matrix. The large gap in the data reflects the fact that is hard to find structured (non-random) objects.

that, when we increase the dimension of the matrices, we obtain a wider gap within the intersection point of the two curves and the expected BDM value, which corresponds to the expected algorithmic complexity of randomly generated matrices. However, we also increase the number of structured matrices, ultimately producing a richer and more interesting evolution space.

8.3.6 Chasing Biological and Synthetic Dynamic Nets

Chasing A Biological Network

I now set as target the adjacency matrix of a biological network corresponding to the topology of an ERBB signaling network [45]. The network is involved in responses ranging from cell division, death, motility, and adhesion and when dysregulated it has been found to be strongly related to cancer [77, 61].

As one of my main hypotheses is that algorithmic probability is a better model for explaining biological diversity, it is important to explore whether *naturally occurring* structures are more likely to be produced under the BDM strategy than the uniform strategy, which is equivalent to showing them evolving faster.

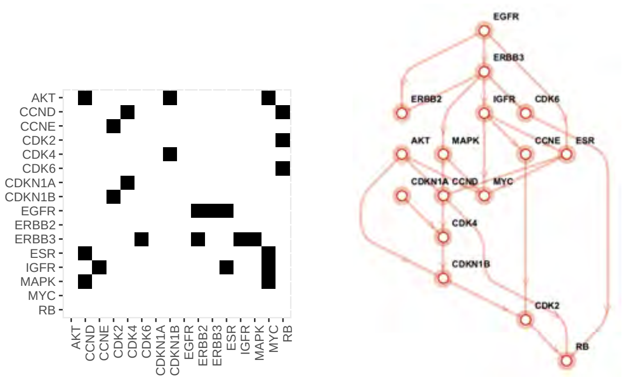

Figure 8.8: Adjacency matrix (left) of an ERBB signalling network (right).

The binary target matrix is shown in Figure 8.8 and it has a BDM of 349.91 bits. For the first experiment, I generated 50 random matrices that were evolved using 1-bit shift mutations for the Uniform and BDM distributions, without repetitions. The BDM of the matrix is at the right of the intersection point inferred by the cubic models shown in Figure 8.7. Therefore we predict a slow-down. The results obtained are shown in the table 8.3.

Table 8.3: Results obtained for the ERBB Network

Strategy	Shifts	Average	SE	Extinctions
Uniform	1	1222.62	23.22	0
BDM	1	1721.86	56.88	0

As the results show, we obtained a slow-down of 0.71, without extinctions. However, as mentioned above, the BDM of the target matrix is relatively high, so this result is consistent with the previous experiments. However, for this case the strategy can be significantly improved.

The Case for Localized Mutations and Modularity

As previously mentioned in the proposition 8.3, the main causes of slowdown under the BDM distribution are maladaptive persistent structures. These structures will negatively impact the evolution speed in factorial order relative to the size of the state space. One direct way to reduce the size set of possible mutations is to reduce the size of the matrices we are evolving. However, doing so will reduce the number of interesting objects we can evolve towards too. Another way to accomplish the objective while using the same heuristic is to rely on *localized* (or *modular*) mutations. That is, we force the mutation to take place on a submatrix of the input matrix.

The way we implement the stated change is by adding a single step in our evolution dynamics: at each iteration, we will randomly draw, with uniform probability, one submatrix of size 4×4 out of the set of adjacent submatrices that compose the input matrix, with no overlap, and force the mutation to be there by computing the probability distribution over all the matrices that contain the bit-shift only at the chosen place. We will call this method the *local BDM* method.

It is important to note that, within 1-bit shifts (point mutations), the space of total possible mutations remains the same when compared to the uniform and BDM strategies. Furthermore, the behavior of the uniform strategy would remain unchanged if the extra step is applied using the Uniform distribution.

We repeated the experiment shown in the table 8.3 with the addition of the local BDM strategy and the same 50 random initial matrices. Its results are shown in the table 8.4. As we can see from the results obtained, local BDM obtains a statistically significant speed-up of 1.25 when compared to the uniform.

Table 8.4: Results obtained for the ERBB Network

Strategy	Shifts	Average	SE	Extinctions
Uniform	1	1222.62	23.22	0
BDM	1	1721.86	56.88	0
Local BDM	1	979	25.94	0

One potential explanation of why we failed to obtain speed-up for the network with the BDM strategy is that, as an approximation to K, the model depends on finding global algorithmic structures, while the sample is based on a substructure which might not have enough information about the underlying structures we hypothesize govern the natural world and make possible scientific models and predictions.

However, biology evolves modular systems [58], such as genes and cells, that in turn build building blocks such as proteins and tissues. Therefore, local algorithmic mutation is a better model. This is a good place to recall that local BDM was devised as a natural solution to the problem presented by maladaptive persistent structures in global algorithmic mutation. Which also means that this type of modularity can be evolved by itself given that it provides an evolutionary advantage, as our results demonstrate.

I will further explore the relationship between BDM and local BDM within the context of global structures in the next section. My current setup is not optimal for further experimentation in *biological* and local structured matrices, as the computational resources required to build the probability distribution for each instance grows in quadratic order relative to matrix size.

Chasing Synthetic Evolving Networks

The aim of the next set of experiments was to follow, or *chase*, the evolution of a moving target using our evolutionary strategies. In this case, I chased 4 different *dynamical networks*: the ZK graphs [80], K-ary trees, an evolving N-star graph and a star-to-path graph dynamic transition artificially created for this project (see Appendix for code). These dynamical networks are families of directed labeled graphs that evolve over time using a deterministic algorithm, some of which display interesting graph-theoretic and entropy-fooling properties [80]. As the evolution dynamics of these graphs are fully deterministic, we expected BDM to be significantly faster than the other two evolutionary strategies, uniform probability and local BDM.

I chased these dynamics in the following way: Let S_0, S_1, $\ldots$, S_n, $\ldots$ be the stages of the system we are chasing. Then the initial state S_0 was represented by a random matrix and, for each evolution $S_i \mapsto S_{i+1}$, the input was defined as the adjacency matrix corresponding to S_i, while the target was set as the adjacency matrix for S_{i+1}. In order to normalize the matrix size, I defined the networks as always containing the same number of nodes (16 for 16×16 matrices). I followed each dynamic until the corresponding stage could not be defined in 16 nodes.

The results which were obtained, starting from 100 random graphs and 100 different evolution paths at each stage, are shown in Figure 8.9. It is important to note that, since the graphs were directed, the matrices used were non-symmetrical.

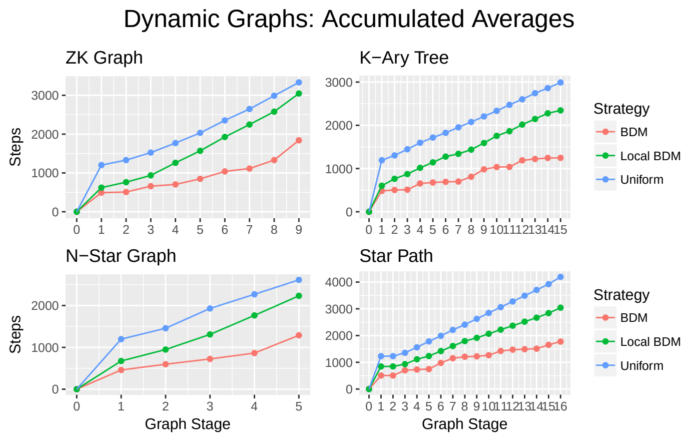

Figure 8.9: Each point of the graph is composed of the average accumulated time from 100 samples. All evolutions were carried out with 1-bit shifts (mutations).

From the results we can see that local BDM consistently outperformed the uniform probability evolution, but the BDM strategy was the faster by a significant margin. The results are as expected and confirm our hypothesis: *uniform evolution cannot detect any underlying algorithmic cause of evolution, while BDM can, inducing a faster overall evolution.* Local BDM can only detect local regularities, which is good enough to outrun uniform BDM in these cases. However, as the algorithmic regularities are global, local BDM is slower than (global) BDM.

Chapter 9

Conclusions

The Open-Ended Evolution question.

Is undecidability a requirement for OEE? In order to formally prove that undecidability of adaptation is a requirement for OEE, I presented a formal and general mathematical model for adaptation within the framework of computable dynamical systems. This model exhibits universal properties for all computable dynamical systems, including the set of Turing machines. Among other results, I have given formal definitions for open-ended evolution (OEE) and strong open-ended evolution and supported the latter on the basis that it allows us to differentiate between trivial and non-trivial systems.

I have also shown that **decidability imposes universal limits on the growth of complexity in computable systems**, as measured by sophistication, coarse sophistication and busy beaver logical depth. I showed that time dominates the descriptive algorithmic complexity of the states, therefore the complexity of the evolution of a system tightly follows that of natural numbers, implying the existence of non-trivial states but the non-existence of an algorithm for finding these states or any subsequence of them, which makes the computations for harnessing or identifying them undecidable.

Furthermore, as a direct implication of corollary 5.2 and theorem 5.3, the undecidability of adapted states and the unpredictability of the behavior of the system at each state is a requirement for a system to exhibit strong open-ended evolution with respect to the complexity measures known as sophistication, coarse sophistication and busy beaver logical depth, providing rigorous proof that undecidability and irreducibility of future behavior is a requirement for the growth of complexity in the class of computable dynamical systems. I conjecture that these results can be extended to any adequate complexity measure that assigns low complexity to random objects.

Finally, I provided an example of an (incomputable) evolutionary system that exhibits strong OEE and supplied arguments for its adequacy as a model of evolution, which I claim supports my characterization of strong OEE. This model was proposed by Chaitin within the framework of metabiology and introduces the concept of **algorithmic probable mutations** (definition 6.7) and algorithmic probability driven evolution in order to evolve toward non-trivially complex functions. I assert that this model is a good analogue to biological evolution, given that adaptation can be seen as analogue to the problem of finding the values of the busy beaver function, producing complexity and long lasting simplicity as a byproduct. *It is my believe that algorithmic probability driven evolution establishes a new line of research in artificial and biological evolution.*

Finding a good computable approximation to m(s).

The function $m(s)$ is the probability of a randomly chosen Turing machine or, equivalently, any computable process to produce the string s. This probability function is known as the *universal distribution*. A *good* computable approximation for it is required in order to do numerical experiments in algorithmic probability driven evolution.

The Block Decomposition Method is a theoretically sound and robust measure of complexity that connects two of the main branches of information theory, classical and algorithmic. The methods used are scalable in various ways, including native n-dimensional variations of the same measure. The properties and numerical experiments are in alignment with theoretical expectations and represent the only truly different alternative and more accurate measure of algorithmic complexity currently available [86].

Furthermore, by means of CTM, BDM is a very good approximation for the algorithmic complexity of relative small binary matrices. With this measure I was able to define a good approximation to $m(s)$ for matrices of size up to 16×16 and, using the coding theorem 6.4, it allowed us to define a computable approximation to the probable mutations distribution for these matrices within a restricted number of shifts. This distributions were used for the experimental part of this work.

Exploring algorithmic probability driven evolution.

The experiments show that algorithmic probability driven evolution converges faster towards no-trivial complex structures than the alternative: that all mutations have the same probability.

These findings demonstrate that key aspects of life may be better explained with this algorithmic account and that the empirical observation of the rate of biological evolutionary convergence emerges naturally, unlike the alternative—and generally assumed–hypothesis.

By assuming this new approach to explain novelty in evolution, we can answer questions ranging from the apparition of sudden major stages of evolution, the emergence of 'subroutines' in the form of modular persistent structures and the need of an evolving memory carrying information organized in such modules that drive the speed-up in the process of evolution by selection.

The interplay of the evolvability of organisms from the persistence of such structures also explains two opposed phenomena: recurrent explosions of diversity and mass extinctions, phenomena which have occurred during the history of life on earth that have not been satisfactorily explained under the uniform mutation assumption. By taking the informational and computational aspects of life based on modern synthesis to the ultimate and natural consequences, the present approach based on weak assumptions of deterministic dynamic systems could offer a novel framework of algorithmic evolution within which to study both biological and artificial evolution.

Furthermore, the success of obtaining *faster than random* evolution by experimentation **reinforces** the theoretical framework presented during the first half of this dissertation.

9.1 Final Remarks

On the Origin of Non-Trivial Information

While discussing this work with colleagues from different fields I have noticed that most disagreements come from a different understanding of the origin of complexity and randomness. For example, one criticism to the evolutionary model presented in chapter 8 is that it seems too simple for it to convey all the properties I assign to it. However, my model is capable of non-trivial evolution not because of its mechanics, which are indeed simple, but from the underlying approximation to the *universal distribution.* I consider such comments a misunderstanding of the nature of information. The database my system uses took exhaustive amounts of computational resources to be compiled (12 days of calculation with a supercomputer of medium size: $25 \times 86-64$ CPUs running at 2,128 MHz each with 4 GB of memory each [83]).

Like HP, Ω and BB, $m(s)$ is an object that contains an infinite amount of information that can be exploited by computable means if we have access to it. Trivial sets, like the set of natural numbers $\mathbb{N}$, contain *all the answers*, but the problem lies in knowing which of all the binary strings encodes the correct proof of Goldbach's conjecture. Knowing the location of the answer is as hard as knowing the answer itself; this is the essence of undecidability and for this reason no computable object can generate more discoverable information than what it was originally encoded within it. Everything you can discover with a decidable function was set *a priori* during its definition.

Even if you feed this computable system with new information, like the natural numbers (discrete time) or stochastically chosen real numbers (*regular* evolution), a computable function has no way of discerning between trivial or non-trivial information. Otherwise we would find a contradiction to randomness itself. It follows that a system will not evolve non trivial structures over long periods of times unless its being feed non trivial information to begin with, which is what the universal distribution does for algorithmic probability driven evolution.

For a computable dynamical system, the mechanics and its initial conditions become increasingly irrelevant as times goes on. Time (or randomness) will eventually overwhelm what little structure there was at the beginning. However, when the feedback it is receiving comes from an object of high information, then structure (non-trivial information) will prevail in its place. Am I just changing the origin of randomness? In a sense, the origin of randomness is everything, as any amount of information a model can start with is minuscule compared to the amount of the information contained on the evolution of the system itself.

Most scientific research has been centered in models aimed at controlling randomness. As I showed in the first part of my thesis, this approach has inherent restrictions that are, in a way, more stringent and fundamental than conventional, widespread and understood scientific limits such as chaos and the uncertainty principle. The alternative, randomness controlling the evolution of a model, should be seriously looked at, as I have showed this approach has the potential to explain scientific phenomena.

Chapter 10

Bibliography

[1] Adami, C. What is complexity? *BioEssays 24*, 12 (2002), 1085–1094.

[2] Adami, C., and Brown, C. T. Evolutionary learning in the 2D artificial life system avida. In *Proc. Artificial Life IV* (1994), MIT Press, pp. 377–381.

[3] Adams, A., Zenil, H., Davies, P. C., and Walker, S. I. Formal definitions of unbounded evolution and innovation reveal universal mechanisms for open-ended evolution in dynamical systems. *Scientific Reports 7* (2017).

[4] Antunes, L., and Fortnow, L. Sophistication revisited. In *ICALP: Annual International Colloquium on Automata, Languages and Programming* (2003).

[5] Auerbach, J. E., and Bongard, J. C. Environmental influence on the evolution of morphological complexity in machines. *PLoS Computational Biology 10*, 1 (2014).

[6] Bedau. Four puzzles about life. *ARTLIFE: Artificial Life 4* (1998).

[7] Bedau, McCaskill, Packard, Rasmussen, Adami, Green, Ikegami, Kaneko, and Ray. Open problems in artificial life. *ARTLIFE: Artificial Life 6* (2000).

[8] Bennett, C. H. Logical depth and physical complexity. In *The Universal Turing Machine: A Half-Century Survey* (1988), R. Herken, Ed., Oxford University Press, pp. 227–257.

[9] Blondel, V. D., Bournez, O., Koiran, P., and Tsitsiklis, J. N. The stability of saturated linear dynamical systems is undecidable. In *Symposium on Theoretical Aspects of Computer Science (STACS), Lille, France* (Feb 2000), H. R. S. Tison, Ed., vol. 1770 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 479–490.

[10] Bournez, O., Graça, D. S., Pouly, A., and Zhong, N. *The Nature of Computation. Logic, Algorithms, Applications: 9th Conference on Computability in Europe, CiE 2013, Milan, Italy, July 1-5, 2013. Proceedings.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, ch. Computability and Computational Complexity of the Evolution of Nonlinear Dynamical Systems, pp. 12–21.

[11] Brady, A. H. The determination of the value of rado's noncomputable function σ for four-state turing machines. *Mathematics of Computation 40*, 162 (1983), 647–665.

[12] Calude, C. S. *Information and Randomness - An Algorithmic Perspective.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002.

[13] Calude, C. S. *Information and randomness: an algorithmic perspective.* Springer Science & Business Media, 2013.

[14] Calude, C. S., Dinneen, M. J., Shu, C.-K., et al. Computing a glimpse of randomness. *Experimental Mathematics 11*, 3 (2002), 361–370.

[15] Calude, C. S., Hertling, P. H., Khoussainov, B., and Wang, Y. Recursively enumerable reals and Chaitin Ω numbers. *Theoretical Computer Science 255* (2001), 125–149.

[16] Calude, C. S., and Jurgensen, H. Is complexity a source of incompleteness? *Advances in Applied Mathematics 35*, 1 (2005), 1 – 15.

[17] Calude, C. S., and Stay, M. A. Most programs stop quickly or never halt. *Advances in Applied Mathematics 40*, 3 (2008), 295–308.

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

[19] Chaitin, G. J. Information-theoretic limitations of formal systems. *Journal of the ACM 21*, 3 (July 1974), 403–424.

[20] Chaitin, G. J. Algorithmic information theory. In *Encyclopaedia of Statistical Sciences*, vol. 1. Wiley, 1982, pp. 38–41.

[21] Chaitin, G. J. Evolution of mutating software. *Bulletin of the EATCS 97* (2009), 157–164.

[22] Chaitin, G. J. Life as evolving software. In *A Computable Universe: Understanding and Exploring Nature as Computation*, H. Zenil, Ed. World Scientific Publishing Company, 10 2012, ch. 16.

[23] Chaitin, G. J. *Proving Darwin: Making Biology Mathematical.* Vintage, 2013.

[24] Church, A. An unsolvable problem of elementary number theory. *American Journal of Mathematics 58* (1936), 354–363.

[25] Cilibrasi, R., and Vitányi, P. M. Clustering by compression. *IEEE Transactions on Information theory 51*, 4 (2005), 1523–1545.

[26] Cooper, S. B. Emergence as a computability-theoretic phenomenon. *Applied Mathematics and Computation 215*, 4 (2009), 1351–1360.

[27] Copeland, B. J. The church-turing thesis. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., summer 2015 ed. Metaphysics Research Lab, Stanford University, 2015.

[28] Daley, R. Busy beaver sets: Characterizations and applications. *INFCTRL: Information and Computation (formerly Information and Control) 52* (1982), 52–67.

[29] Delahaye, J.-P., and Zenil, H. Numerical evaluation of algorithmic complexity for short strings: A glance into the innermost structure of randomness. *Applied Mathematics and Computation 219*, 1 (2012), 63–77.

[30] Delvenne, J.-C., Kurka, P., and Blondel, V. D. Decidability and universality in symbolic dynamical systems. *Fundam. Inform 74*, 4 (2006), 463–490.

[31] Fredkin, E., and Toffoli, T. Conservative logic. *International Journal of Theoretical Physics 21* (1982), 219–253.

[32] Gardner, M. Mathematical games: The random number omega bids fair to hold the mysteries of the universe. *SCIAM: Scientific American 241* (1979).

[33] Gauvrit, N., Singmann, H., Soler-Toscano, F., and Zenil, H. Algorithmic complexity for psychology: a user-friendly implementation of the coding theorem method. *Behavior research methods 48*, 1 (2016), 314–329.

[34] Gauvrit, N., Soler-Toscano, F., and Zenil, H. Natural scene statistics mediate the perception of image complexity. *Visual Cognition 22*, 8 (2014), 1084–1091.

[35] Gell-Mann, M. What is complexity? remarks on simplicity and complexity by the nobel prize-winning author of the quark and the jaguar. *Complexity 1*, 1 (1995), 16–19.

[36] H. Zenil, F. Soler-Toscano, J.-P. D., and Gauvrit, N. Two-dimensional Kolmogorov complexity and validation of the coding theorem method by compressibility. *PeerJ Computer Science 1:e23* (2015).

[37] Hall, N., Ed. *Exploring chaos: A guide to the new science of disorder.* W. W. Norton & Co., New York, 1991.

[38] Hartl, D. L., Clark, A. G., and Clark, A. G. *Principles of population genetics*, vol. 116. Sinauer associates Sunderland, 1997.

[39] Hernández-Orozco, S., Hernández-Quiroz, F., and Zenil, H. The limits of decidable states on open-ended evolution and emergence. In *Proceedings of the Artificial Life Conference 2016* (2016), MIT Press, pp. 200–2008.

[40] Hernández-Orozco, S., Hernández-Quiroz, F., and Zenil, H. The Limits of Decidable States on Open-Ended Evolution and Emergence. *Artificial Life 24:1* (2018), 56–70.

[41] Hernández-Orozco, S., Zenil, H., and Kiani, N. A. Algorithmically probable mutations reproduce aspects of evolution such as convergence rate, genetic memory, modularity, diversity explosions, and mass extinction. *ArXiv e-prints* (Sept. 2017).

[42] Hutter, M. Algorithmic information theory. *Scholarpedia 2*, 3 (2007), 2519. revision #90953.

[43] Hutter, M., Legg, S., and Vitanyi, P. M. Algorithmic probability. *Scholarpedia 2*, 8 (2007), 2572. revision #151509.

[44] Kempe, V., Gauvrit, N., and Forsyth, D. Structure emerges faster during cultural transmission in children than in adults. *Cognition 136* (2015), 247–254.

[45] Kiani, N. A., and Kaderali, L. Dynamic probabilistic threshold networks to infer signaling pathways from time-course perturbation data. *BMC bioinformatics 15*, 1 (2014), 250.

[46] Kirchherr, W., Li, M., and Vitányi, P. The miraculous universal distribution. *The Mathematical Intelligencer 19*, 4 (1997), 7–15.

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

[48] Koppel, M. Structure. In *The Universal Turing Machine: A Half-Century Survey* (1988), R. Herken, Ed., Oxford University Press, pp. 435–452.

[49] Kraft, L. G. *A device for quantizing, grouping, and coding amplitude-modulated pulses.* PhD thesis, Massachusetts Institute of Technology, 1949.

[50] Langton, C. G. Studying artificial life with cellular automata. *Physica D: Nonlinear Phenomena 22*, 1-3 (1986), 120–149.

[51] Lehman, J., and Stanley, K. O. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE* (2008), S. Bullock, J. Noble, R. A. Watson, and M. A. Bedau, Eds., MIT Press, pp. 329–336.

[52] LEVIN, L. A. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii 10*, 3 (1974), 30–35.

[53] LI, M., AND VITÁNYI, P. *An introduction to Kolmogorov complexity and its applications*, 2nd ed. Springer, 1997.

[54] LINDGREN, K. Evolutionary phenomena in simple dynamics. In *Artificial Life II*, C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds. Addison-Wesley, Redwood City, CA, 1992, pp. 295–312.

[55] MARGOLUS, N. Physics-like models of computation. *Physica D 10* (1984), 81–95.
Discussion of reversible cellular automata illustrated by an implementation of Fredkin's Billiard-Ball model of computation.

[56] MCMILLAN, B. Two inequalities implied by unique decipherability. *IRE Transactions on Information Theory 2*, 4 (1956), 115–116.

[57] MEISS, J. Dynamical systems. *Scholarpedia 2*, 2 (2007), 1629. revision #121407.

[58] MITRA, K., CARVUNIS, A.-R., RAMESH, S. K., AND IDEKER, T. Integrative approaches for finding modular structure in biological networks. *Nature reviews. Genetics 14*, 10 (2013), 719.

[59] MOORE, C. Generalized shifts: Unpredictability and undecidability in dynamical systems. *Nonlinearity 4*, 2 (1991), 199–230.

[60] O'CONNOR, T., AND WONG, H. Y. Emergent properties. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., summer 2015 ed. Metaphysics Research Lab, Stanford University, 2015.

[61] OLAYIOYE, M. A., NEVE, R. M., LANE, H. A., AND HYNES, N. E. The ErbB signaling network: receptor heterodimerization in development and cancer. *EMBO J 19*, 3 (2000), 3159–67.

[62] RADO, T. On non-computable functions. *Bell Labs Technical Journal 41*, 3 (1962), 877–884.

[63] Ruiz-Mirazo, K., Peretó, J., and Moreno, A. A universal definition of life: Autonomy and open-ended evolution. *Origins of life and evolution of the biosphere 34*, 3 (2002), 323–346.

[64] Schaffer, J., and Eshelman, L. On crossover as an evolutionary viable strategy. In *Proceedings of the 4th International Conference on Genetic Algorithms* (1991), R. Belew and L. Booker, Eds., Morgan Kaufmann, pp. 61–68.

[65] Schmitt, A. O., and Herzel, H. Estimating the entropy of dna sequences. *Journal of Theoretical Biology 188*, 3 (1997), 369 – 377.

[66] Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal 27* (July / Oct. 1948), 379–423, 623–656.

[67] Soler-Toscano, F., Zenil, H., Delahaye, J.-P., and Gauvrit, N. Correspondence and independence of numerical evaluations of algorithmic information measures. *Computability 2*, 2 (2013), 125–140.

[68] Soler-Toscano, F., Zenil, H., Delahaye, J.-P., and Gauvrit, N. Calculating kolmogorov complexity from the output frequency distributions of small turing machines. *PLoS One 9*, 5 (2014), e96223.

[69] Solomonof. The universal distribution and machine learning. *COMPJ: The Computer Journal 46* (2003).

[70] Solomonoff, R. J. A formal theory of inductive inference. *Information and Control 7* (1964), 1–22, 224–254.

[71] Soros, L. B., and Stanley, K. O. Identifying necessary conditions for open-ended evolution through the artificial life world of chromaria. In *Fourteenth International Conference on the Synthesis and Simulation of Living Systems (ALIFE 14)* (2014), MIT Press.

[72] Standish, R. K. Open-ended artificial evolution. *International Journal of Computational Intelligence and Applications 3*, 2 (2003), 167–175.

[73] Taylor, T. Requirements for open-ended evolution in natural and artificial systems. *CoRR abs/1507.07403* (2015).

[74] Turing, A. M. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society 42* (1936), 230–265.

[75] Wilson, R., and Keil, F. *The MIT Encyclopedia of the Cognitive Sciences*. Bradford Books. MIT Press, 2001, p. 37.

[76] Wolfram, S. *A New Kind of Science*. Wolfram Media Inc., 2002.

[77] Yarden, Y., and Sliwkowski, M. Untangling the ErbB signalling network. *Nat Rev Mol Cell Biol. 2*, 2 (2001), 127–37.

[78] Zenil, H. Small data matters, correlation versus causation and algorithmic data analytics. *Predictability in the world: philosophy and science in the complex world of Big Data, Springer Verlag (forthcoming)* (2013).

[79] Zenil, H., Gershenson, C., Marshall, J. A. R., and Rosenblueth, D. A. Life as thermodynamic evidence of algorithmic structure in natural environments. *Entropy 14*, 11 (2012).

[80] Zenil, H., Kiani, N. A., and Tegnér, J. Low-algorithmic-complexity entropy-deceiving graphs. *Physical Review E 96*, 1 (2017), 012308.

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

[82] Zenil, H., and Marshall, J. Some aspects of computation essential to evolution and life. *Ubiquity April 2013* (2013), 1–16.

[83] Zenil, H., Soler-Toscano, F., Delahaye, J.-P., and Gauvrit, N. Two-dimensional kolmogorov complexity and an empirical validation of the coding theorem method by compressibility. *PeerJ Computer Science 1* (2015), e23.

[84] Zenil, H., Soler-Toscano, F., Dingle, K., and Louis, A. A. 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* (2014), 341–358.

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

[86] Zenil, H., Soler-Toscano, F., Kiani, N. A., Hernández-Orozco, S., and Rueda-Toicen, A. A decomposition method for global evaluation of shannon entropy and local estimations of algorithmic complexity. *arXiv preprint arXiv:1609.00110* (2016).

[87] Ziv, J., and Lempel, A. A universal algorithm for sequential data compression. *IEEE Transactions on information theory 23*, 3 (1977), 337–343.

[88] Ziv, J., and Lempel, A. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory 24*, 5 (1978), 530–536.