Supporting Information
of the article:

Assembly Theory is an approximation to algorithmic complexity based on LZ compression that does not explain selection or evolution

Felipe S. Abrahão[1,2,3], Santiago Hernández-Orozco[1], Narsis A. Kiani[4,5], Jesper Tegnér[6], Hector Zenil[1,5,7,8,9*¤]

1 Oxford Immune Algorithmics, Oxford University Innovation, Oxford, U.K.
2 Center for Logic, Epistemology and the History of Science, University of Campinas (UNICAMP), Brazil.
3 DEXL, National Laboratory for Scientific Computing (LNCC), Brazil.
4 Department of Oncology-Pathology, Center for Molecular Medicine, Karolinska Institutet, Sweden.
5 Algorithmic Dynamics Lab, Center for Molecular Medicine, Karolinska Institutet, Sweden.
6 Living Systems Lab, King Abdullah University of Science and Technology, Thuwal, Kingdom of Saudi Arabia.
7 The Alan Turing Institute, British Library, London, U.K.
8 School of Biomedical Engineering and Imaging Sciences, King's College London, U.K.
9 King's Institute for Artificial Intelligence, King's College London, U.K.

¤Current Address: School of Biomedical Engineering and Imaging Sciences, King's College London, U.K.
* hector.zenil@kcl.ac.uk

A S1 Appendix

A.1 Mathematical framework

A.1.1 Algorithmic information theory

The (unconditional prefix)[1] *algorithmic complexity* of a finite string w, denoted by $\mathbf{K}(w)$, is the length of the shortest program $w^* \in \mathbf{L_U}$ such that $\mathbf{U}(w^*) = w$ and the length $|x^*| = \mathbf{K}(x)$ of program x^* is minimum, where $\mathbf{L_U}$ denote any (prefix-free or self-delimiting) universal programming language running on a machine $\mathbf{U}$.

Let $\langle x \rangle$ denote an encoding of an object x so that it can be recursively extended to $\langle \cdot, \ldots, \cdot \rangle$ in order to represent the encoding of n-tuples.

From the *algorithmic coding theorem* [1–4] we have it that

$$\begin{aligned} \mathbf{K}(x) = \\ -\log\left(\sum_{\mathbf{U}(p)=x} \frac{1}{2^{|p|}}\right) \pm \mathbf{O}(1) = \\ -\log(\mathbf{m}(x)) \pm \mathbf{O}(1)\ , \end{aligned} \tag{1}$$

holds, where:

[1] In this paper, we will normally refer to prefix algorithmic complexity as just algorithmic complexity.

- p denotes a program running on machine $\mathbf{U}$ such that it returns $\mathbf{U}(p) = x$ as output, where $|p|$ is the length of the program p;
- $\mathbf{m}(\cdot)$ is a maximal computably enumerable semimeasure;
- $\mathbf{P_U}[\text{“event } x \text{ occurs”}] = \sum_{\mathbf{U}(p)=x} 2^{-|p|}$ denotes the *universal a priori probability* of an arbitrary event.

Notice that $\mathbf{P_U}$ can be understood as the probability of randomly generating (by an i.i.d. stochastic process) a prefix-free (or self-delimiting) program that outputs x. A computably enumerable semimeasure $\mathbf{m}(\cdot)$ is said to be *maximal* if, for any other computably enumerable semimeasure $\mu(\cdot)$ with domain defined for all possible encoded objects, where $\sum_{x\in\{0,1\}^*} \mu(x) \leq 1$, there is a constant $C > 0$ (which does not depend on x) such that, for every encoded object x, $\mathbf{m}(x) \geq C\,\mu(x)$. Also note that the algorithmic coding theorem applies analogously to the conditional algorithmic complexity $\mathbf{K}(z\,|w)$.

We also have it that $2^{-\mathbf{K}(x)}$ is called the *algorithmic probability* of x.

One of the basic results in AIT [1–4] is that *no computable* measure (or semimeasure) $\mu'(\cdot)$ of the infinite discrete space of all encodable finite objects can be maximal, unlike the computably enumerable semimeasure $\mathbf{m}(\cdot)$ (or, equivalently, the algorithmic probability) which is maximal, and therefore is guaranteed to eventually dominate any such a $\mu'(\cdot)$.

A.1.2 Assembly theory

As in [5], let (Γ, ϕ) denote either an *assembly space* (AS), or an *assembly subspace*. In the interests of simplifying notation, one can refer to (Γ, ϕ) simply as Γ where the edge-labellings given by ϕ are not relevant. From [5], we have it that $c_\Gamma(x)$ denotes the *assembly index* of the object x in the assembly space Γ.

Let $\mathcal{S} = (\mathbf{\Gamma}, \mathbf{\Phi}, \mathcal{F})$ be an *infinite assembly space*, where every assembly space $\Gamma \in \mathbf{\Gamma}$ is finite, $\mathbf{\Phi}$ is the set of the corresponding edge-labelling maps ϕ_Γ of each Γ, and $\mathcal{F} = (f_1, \ldots, f_n, \ldots)$ is the infinite sequence of embeddings (in which each embedding is also an *assembly map* as in [5]) that ends up generating $\mathcal{S}$. That is, each $f_i\colon \{\Gamma_i\} \subseteq \mathbf{\Gamma} \to \{\Gamma_{i+1}\} \subseteq \mathbf{\Gamma}$ is a particular type of assembly map that embeds a single assembly subspace into a larger assembly subspace so that the resulting sequence of nested assembly subspaces defines a total order $\preceq_\mathcal{S}$, where

$$(\Gamma_i, \phi_{\Gamma_i}) \preceq_\mathcal{S} \left(\Gamma_{i+1}, \phi_{\Gamma_{i+1}}\right) \textit{ iff } f_i(\Gamma_i) = \Gamma_{i+1} \ .$$

In other words, $\mathcal{S}$ is the nested family of all possible finite assembly spaces from the same basis set (i.e., the root vertex that represents the set of all basic building blocks) such that $\mathcal{S}$ is infinite and enumerable by a program.[2]

The source vertex x of the edge $e \in E(\Gamma)$ is given by the function $s_\Gamma(e) = x$. The target vertex y of the edge $e \in E(\Gamma)$ is given by the function $t_\Gamma(e) = y$. Let $\gamma = z \ldots y$ denote an arbitrary path from $z \in B_\mathcal{S}$ to some terminal $y \in V(\mathcal{S})$ in $\mathcal{S}$, where $B_\mathcal{S}$ is the *basis set* (i.e., the finite set of basic building blocks) of $\mathcal{S}$ and $V(\mathcal{S})$ is the set of vertices of $\mathcal{S}$. Let γ_x denote a rooted path from some $z \in B_\mathcal{S}$ to the object $x \in V(\mathcal{S})$. All of these apply analogously to Γ instead of $\mathcal{S}$.

[2] Note that an alternative proof of the results presented in this paper can be achieved just with the (more general) assumption that the assembly space may be finite but only needs to be sufficiently large in comparison to the deceiving program in Lemma 1.

As in [5], let ${\Gamma^*}_x$ denote one of the *minimum rooted assembly subspaces* of Γ from which the *assembly index*

$$\begin{array}{rccc} c_\Gamma\colon & \Gamma \subset \mathcal{S} & \to & \mathbb{N} \\ & x \in V(\Gamma) & \mapsto & c_\Gamma(x) \end{array}$$

calculates the augmented cardinality $c_\Gamma(x)$, i.e., the number of vertices in a minimal rooted assembly subspace, except for those in the basis set B_Γ. The *longest* rooted paths $\gamma_x^{\max}$ of ${\Gamma^*}_x$ end in the object/vertex $x \in V(\Gamma)$, i.e., x is the terminal vertex of the path $\gamma_x^{\max}$ or, equivalently, x is the target vertex of the latest oriented edge in $\gamma_x^{\max}$. The *shortest* rooted paths $\gamma_x^{\min}$ of ${\Gamma^*}_x$ end in the object/vertex $x \in V(\Gamma)$, i.e., x is the terminal vertex of the path $\gamma_x^{\min}$ or, equivalently, x is the target vertex of the latest oriented edge in $\gamma_x^{\min}$.

Let $A\#\colon \mathbf{X} \to \mathbb{N}$ be a function from the set $\mathbf{X}$ of possible ensembles (composed of objects) into the set of natural numbers. As in [6], this function defines the *assembly number* $A\#(X)$ of an ensemble X such that

$$\begin{array}{rccc} A\#\colon & \mathbf{X} & \to & \mathbb{N} \\ & X & \mapsto & A\#(X) = A(N_T, N, (n_1, \ldots, n_N), (a_1, \ldots, a_N)) \end{array} , \tag{2}$$

where:

- N_T is the total number of objects in the ensemble X;
- N is the total number of unique objects in the ensemble N_T;
- each n_i corresponds to the copy number of the (unique) object x_i with $1 \leq i \leq N$;
- $a_i = c_\Gamma(x_i)$, where Γ is the assembly space that contains the object x_i;
- $A(\cdots)$ is the *assembly of the ensemble* X calculated by

$$A(N_T, N, (n_1, \ldots, n_N), (a_1, \ldots, a_N)) = \sum_{i=1}^{N} e^{a_i} \left(\frac{n_i - 1}{N_T} \right) .$$

According to the assumptions and claims in [5–7] (as also discussed in [8]), notice that the values of N_T, N, n_i, a_i are retrievable by an effective procedure from a given (finite) ensemble X, computational methods which are proposed by Assembly Theory [5–7]. Thus, function $A\#$ is a recursive function (i.e., a function that an algorithm can calculate) given a finite ensemble X, a property which the authors of [5–7] falsely claim to be one of the advantages of Assembly Theory over algorithmic complexity, because its methods were computable already to begin with, so there was no room for uncomputability claims.

Let $\langle V(\Gamma) \rangle$ be an encoding of the set $V(\Gamma)$ of vertices of the assembly space (or subspace) Γ. Notice that the encoding formalism and the ordering of the vertices belonging to Γ are arbitrary. The only condition is that once the encoding method is (arbitrarily) chosen, it is fixed and remains the same for any possible assembly space (or subspace) Γ. Analogously, let $\langle \gamma \rangle$ denote an arbitrarily fixed encoding of a sequence/path γ composed of contiguous oriented edges together with the starting vertex/object in the basis set $B_\Gamma \subseteq B_\mathcal{S}$, the sequence which defines the path γ in Γ.

We have it that the (prefix) *algorithmic complexity*, denoted by $\mathbf{K}(x)$, is the length of the shortest prefix-free (or self-delimiting) program that outputs the encoded object x when this program is run using a (universal) programming language.

In accordance with the claims made in [5–7, 9, 10], note that the computability and feasibility of Assembly Theory's methods directly imply the existence of algorithms that

can enumerate the assembly spaces. Given any set of (physical, chemical, and/or biological) rules for assembling molecules that Assembly Theory arbitrarily chooses to be based on, one has it that for any particular assembly (sub)space resulting in a molecule, there is a corresponding algorithm for which one can apply the methods described in [5, 7, 9] to calculate the pathway probability, and so on. This existence is trivially guaranteed to hold as long as the calculation of the assembly index is recursive from a given assembly space, which is already an assumed condition in [5, 7, 9]; moreover, any upper bound for the computational resources necessary to compute the assembly index straightforwardly implies the existence of an upper bound for the computational resources necessary for a program to run. For every permitted generative process that can assemble objects and thereby build another object, there is an algorithm that performs this process. Conversely, for every object, one has it that there is a corresponding generative process allowed by Assembly Theory, a process which is effected by an algorithm. Therefore, according to the assumptions in [5–7, 9, 10], there is a formal theory $\mathbf{F}$ that contains Assembly Theory and all the procedures effected by algorithms, including the chosen method for calculating the assembly index of an object, the program that decides whether or not the criteria for building the assembly spaces are met, and so on.

A.2 Assembly theory is an approximation to algorithmic complexity

One can demonstrate in Lemma 1 and Corollary 2 that the assembly index $c_{\Gamma}(y)$ (or the assembly number $A\#(X)$) is an approximation to algorithmic complexity $\mathbf{K}(y)$ (or, respectively, $\mathbf{K}(X)$) that in general *overestimates* its value as much as desired.

As in [8], the main route to achieving the following theoretical results is to construct a randomly generated program that receives a formal theory (which contains all the procedures and statistical criteria in Assembly Theory) as input. Then, it searches for an object y in an assembly space with a sufficiently high assembly index $c_{\Gamma}(y)$ so as to overcome any arbitrarily chosen error margin k between $c_{\Gamma}(y)$ and the algorithmic complexity $\mathbf{K}(y)$.

Lemma 1. *Let $\mathcal{S}$ be enumerable by an algorithm. Let $\mathbf{F}$ be an arbitrary formal theory that contains Assembly Theory, including all the decidable procedures of the chosen method for calculating the assembly index of an object for a nested subspace of $\mathcal{S}$, and the program that decides whether or not the criteria for building the assembly spaces are met. Let $k \in \mathbb{N}$ be an arbitrarily large natural number. Then, there are $\Gamma \subset \mathcal{S}$ and $y \in V(\Gamma)$ such that*

$$\mathbf{K}(y) + k \leq c_{\Gamma}(y) \ , \tag{3}$$

where the function $c_{\Gamma}(y)$ gives the assembly index of the object y in the assembly space Γ (or $\mathcal{S}$).

Proof. Let p be a bit string that represents an algorithm that receives $\mathbf{F}$ and k as inputs. Then, it calculates $|p| + |\mathbf{F}| + \mathbf{O}(\log_2(k)) + k$ and enumerates $\mathcal{S}$ while calculating $c_{\Gamma}(x)$ of the object (or vertex) $x \in V(\Gamma) \subset V(\mathcal{S})$ at each step of this enumeration. Finally, the algorithm returns the first object $y \in V(\mathcal{S})$ for which

$$|p| + |\mathbf{F}| + \mathbf{O}(\log_2(k)) + k + \mathbf{O}(1) \leq c_{\Gamma}(y) \tag{4}$$

holds. In order to demonstrate that p always halts, just note that $\mathcal{S}$ is infinite computably enumerable. Also, for any value of $c_{\Gamma'}(z)$ for some $z \in V(\Gamma') \subset V(\mathcal{S})$, there is only a finite number of minimum rooted assembly subspaces (starting on any object in $B_{\mathcal{S}}$ and ending on z) whose augmented cardinality is $c_{\Gamma'}(z)$, where $B_{\mathcal{S}}$ is the

basis (i.e., the finite set of basic building blocks [7]) of $\mathcal{S}$. This implies that there is an infinite number of distinct values of $c_{\Gamma'}(z)$. Now, let $\mathrm{p}_y = \langle k, \mathbf{F}, p\rangle$. Finally, from Eq (4) and basic properties in AIT, we have it that

$$\mathbf{K}(y) + k \leq |\mathrm{p}_y| + k + \mathbf{O}(1) \leq |p| + |\mathbf{F}| + \mathbf{O}(\log_2(k)) + k + \mathbf{O}(1) \leq c_\Gamma(y) \quad (5)$$

holds for some sufficiently large k. □

From this proof of Lemma 1, one can directly conclude that:

Corollary 2. *There are infinitely many $\Gamma \subset \mathcal{S}$ for which Eq (3) in Lemma 1 holds.*

Proof. From the proof of Lemma 1 we know that there are infinitely many distinct values of $c_{\Gamma'}(z)$. Let p_2 be a slight variation of program p in the proof of Lemma 1 such that, in addition to $\mathbf{F}$ and k, it also receives a natural number $i \in \mathbb{N}$ as input. Then, it enumerates $\mathcal{S}$ while calculating $c_\Gamma(x)$ of the object (or vertex) $x \in V(\Gamma) \subset V(\mathcal{S})$ at each step of this enumeration. In the last step, program p_2 returns the i-th first object $y \in V(\mathcal{S})$ for which

$$|p| + |\mathbf{F}| + \mathbf{O}(\log_2(k)) + k + \mathbf{O}(\log_2(i)) + i + \mathbf{O}(1) \leq c_\Gamma(y) \quad (6)$$

holds. (Notice that Eq (6) differs from Eq (4)). Then, for each $i > 0$, one will obtain a distinct object y such that

$$\mathbf{K}(y) + k \leq |p| + |\mathbf{F}| + \mathbf{O}(\log_2(k)) + k + \mathbf{O}(\log_2(i)) + i + \mathbf{O}(1) \leq c_\Gamma(y) \quad (7)$$

holds. □

A.2.1 Assembly number and algorithmic complexity

Now we can extend Lemma 1 to the assembly number $A\#$, a value which is defined upon an ensemble X. The main idea of Lemma 3 is that the class of all possible ensembles of objects that assembly spaces can possibly cover is also enumerable by an algorithm, based on the fact that assembly spaces are enumerable by an algorithm. In this manner, as done in Lemma 1, one can make the error margins (in comparison to algorithmic complexity) as large as one wishes.

Lemma 3. *Let $\mathcal{S}$ be enumerable by an algorithm. Let $\mathbf{F}$ be an arbitrary formal theory that contains Assembly Theory, including: all the decidable procedures of the chosen method for calculating the assembly index c_Γ of an object for a nested subspace of $\mathcal{S}$ and the assembly number $A\#$ of a given ensemble X; the program that decides whether or not the criteria for building the assembly spaces are met; and the program that decides whether or not the criteria for building ensembles of objects are met. Let $k \in \mathbb{N}$ be an arbitrarily large natural number. Then, there is $y \in \mathbf{X}$ such that*

$$\mathbf{K}(X) + k \leq A\#(X) \ , \quad (8)$$

where the function $A\#\colon \mathbf{X} \to \mathbb{N}$ gives the assembly number of the ensemble $X \in \mathbf{X}$.

Proof. Remember the definition of assembly number $A\#$ such that $A\#(X) = \sum_{i=1}^{N} e^{a_i}\left(\frac{n_i - 1}{N_T}\right)$, where the parameters N_T, N, n_i, a_i are computably retrievable from a given ensemble X. Since $\mathcal{S}$ is infinite computably enumerable and the theory F can always decide whether or not an arbitrary set Y is an ensemble from which $A\#$ can be calculated, one has it that the set $\mathbf{X}$ (of all possible finite ensembles) is infinite computably enumerable. From here on, the proof follows analogously to the proof of Lemma 1. □

In the same manner as in Corollary 2 with respect to Lemma 1, one can obtain the following corollary:

Corollary 4. *There are infinitely many $X \in \mathbf{X}$ for which Eq (8) in Lemma 3 holds.*

Proof. Analogous to the proof of Corollary 2. □

The reader is invited to note that the latter Lemma 3 and Corollary 4 can also be generalised for any recursive function f defined on the basis of the assembly index and on an arbitrary observable (for which Assembly Theory, in particular, would include a formal criterion to decide whether or not function f can be calculated from this observable). For example, it applies to the arbitrary function $A''\#(X)$ from Section A.3.4.

A.3 Assembly theory is an encoding/compression scheme subsumed into algorithmic information theory

A.3.1 The assembly index defines an encoding scheme

The following results show that a minimal rooted assembly (sub)space, from which assembly index $c_\Gamma(y)$ is calculated, constitutes a parameter that can be employed to encode the minimal assembling process of an object y. More formally, Theorem 5 demonstrates that any arbitrarily chosen encoding of the set of vertices (in any arbitrary ordering) of a minimal rooted assembly space is sufficient to encode the highest-assembly-index object assembled across this space. In other words, the sheer collection of objects in a minimal rooted assembly subspace carries enough information to allow one to always pick the object with the highest assembly index, even though the ordering of the objects in this collection may be completely arbitrary in the first place. This demonstrates that AT's assembly index calculation method is one of the possible methods for encoding objects.

Theorem 5. *Let $\mathcal{S}$ be enumerable by an algorithm. Let $\mathbf{F}$ be an arbitrary formal theory that contains Assembly Theory, including all the decidable procedures of the chosen method for calculating the assembly index of an object for a nested subspace of $\mathcal{S}$, and the program that decides whether or not the criteria for building the assembly spaces are met. Let $\Gamma \subset \mathcal{S}$ with $y \in V(\Gamma)$ be arbitrary. Then, one has it that $\langle V({\Gamma^*}_y)\rangle$ encodes the object y such that*

$$\mathbf{K}(y) \leq \mathbf{K}(\langle V({\Gamma^*}_y)\rangle) + \mathbf{O}(1) \tag{9}$$

holds, where: $c_\Gamma(y)$ gives the assembly index of the object y in the assembly space Γ (or $\mathcal{S}$); ${\Gamma^*}_y$ is the minimum rooted assembly subspace of Γ from which the assembly index $c_\Gamma(y)$ is calculated; and $\langle V\rangle$ is an arbitrarily fixed encoding of a set V of vertices.

Proof. Remember that ${\Gamma^*}_y$ denotes the minimum rooted assembly subspace of Γ from which the assembly index calculates the augmented cardinality that defines the value $c_\Gamma(y)$. Notice that since ${\Gamma^*}_y$ is minimal, then there is a simple algorithm that can always calculate the integer value of $c_\Gamma(y)$ given $\mathbf{K}(\langle V({\Gamma^*}_y)\rangle)$ as input. Remember that, from our definitions and conditions in Section A.1, $\langle V({\Gamma^*}_y)\rangle$ denotes an encoding of the set of vertices of ${\Gamma^*}_y$, an encoding method which is arbitrary, previously chosen, and does not depend on the choice of the assembly (sub)space or object. Now, one can exploit the computability of Assembly Theory and construct a program p_3 that receives $\mathbf{F}$ and $\langle V({\Gamma^*}_y)\rangle$ as inputs. By using $\mathbf{F}$ to decide whether or not an arbitrary Γ' satisfies the criteria of being an assembly subspace, program p_3 enumerates the finite set $\mathbf{\Gamma}(V({\Gamma^*}_y))$ of all possible assembly subspaces whose set of vertices is exactly $V({\Gamma^*}_y)$. Next, using $\mathbf{F}$ to decide whether or not an arbitrary Γ' satisfies the criteria of being a

minimum rooted assembly subspace whose longest path ends in the same last vertex of Γ', program p_3 searches in $\mathbf{\Gamma}\left(V\left({\Gamma^*}_y\right)\right)$ for the first assembly subspace Γ'' that is minimal. Finally, it returns the object $y' \in V\left(\Gamma''\right)$ at which the longest rooted paths in Γ'' end (i.e., y' is the terminal vertex of these longest rooted paths). In order to achieve the desired proof, one has yet to show that the object built at the end of the longest paths in the minimum rooted assembly subspaces is unique and that $y' = y$. To this end, one can exploit the minimality of ${\Gamma^*}_y$. First, suppose there is a minimum $\Gamma_2 \in \mathbf{\Gamma}\left(V\left({\Gamma^*}_y\right)\right)$ that ends at the object z such that $\Gamma_2 \neq \Gamma''$ and $z \neq y'$. Then, by construction of the program p_3, one would have it that y' belongs to at least one of the rooted paths in Γ_2. Hence, there would be a rooted path that leads to y' that is shorter than the one that leads to z because by hypothesis one has it that the path that leads to z is the longest one; and if there were another terminal vertex w in Γ_2 whose rooted path has the same length as that of z, then Γ_2 would not be a minimum subspace. As a consequence, there would be a rooted assembly subspace $\Gamma^*_{2y'} \subseteq \Gamma_2$ whose augmented cardinality is smaller than $c_{\Gamma''}\left(y'\right)$. However, by construction, we have it that $c_{\Gamma''}\left(y'\right)$ is minimal, which leads to a contradiction. Therefore, since by hypothesis we have it that ${\Gamma^*}_y$ is minimal and ${\Gamma^*}_y \in \mathbf{\Gamma}\left(V\left({\Gamma^*}_y\right)\right)$, we conclude that either $\Gamma_2 = \Gamma'' = {\Gamma^*}_y$ or $z = y' = y$, and hence that, in any case, program p_3 produces y as output. Finally, the proof of Eq (9) follows from basic inequalities in AIT. □

Notice that $\langle V\left({\Gamma^*}_y\right)\rangle$ is only built to encode the membership relation for the set, and *does not need to carry any a priori information about its internal structure* that results in the assembly space from which it only takes the set of vertices. Only by applying a third-party decoding algorithm, such as program p_3 in the proof of Theorem 5, does the final object y become uniquely retrievable.

One can demonstrate that from encoding one[3] of the longest[4] paths in a minimal rooted assembly subspace, one can always retrieve one of the minimum rooted assembly subspaces, and hence the terminal vertex that corresponds to the unique object to be assembled:

Corollary 6. *Let $\mathcal{S}$ be enumerable by an algorithm. Let $\mathbf{F}$ be an arbitrary formal theory that contains Assembly Theory, including all the decidable procedures of the chosen method for calculating the assembly index of an object for a nested subspace of $\mathcal{S}$, and the program that decides whether or not the criteria for building the assembly spaces are met. Let $\Gamma \subset \mathcal{S}$ with $y \in V\left(\Gamma\right)$ and $\gamma_y^{\max}$ be arbitrary, where $\gamma_y^{\max}$ is one of the longest rooted paths in ${\Gamma^*}_y$ that has y as its terminal vertex and ${\Gamma^*}_y$ is the minimum rooted assembly subspace of Γ from which the assembly index $c_\Gamma\left(y\right)$ is calculated. Then, one has it that $\left\langle\gamma_y^{\max}\right\rangle$ encodes the object y such that*

$$\mathbf{K}\left(y\right) \leq \mathbf{K}\left(\left\langle\gamma_y^{\max}\right\rangle\right) + \mathbf{O}(1) \tag{10}$$

holds, where $\langle\gamma\rangle$ is any arbitrarily fixed encoding of a sequence γ composed of contiguous oriented edges together with the starting vertex/object in the basis set $B_\Gamma \subseteq B_{\mathcal{S}}$, the sequence which defines the path γ.

Proof. Let p_3 be the program used in the proof of Theorem 5. Let p_4 be a program that receives $\mathbf{F}$, program p_3, and $\left\langle\gamma_y^{\max}\right\rangle$ as inputs. Then, by using $\mathbf{F}$ to decide whether or not an arbitrary Γ' satisfies the criteria of being a minimum rooted assembly subspace,

[3]There might be more than one of these paths. Corollary 6 holds in any case, i.e., it holds for any arbitrarily chosen $\gamma_y^{\max}$ in ${\Gamma^*}_y$.

[4]Notice that a directly analogous version of Corollary 6 holds for $\gamma_y^{\min}$ instead of $\gamma_y^{\max}$. From encoding any of the shortest paths in a minimum rooted assembly subspace, one can always directly retrieve one of the minimum rooted assembly subspaces, and hence the terminal vertex that corresponds to the unique object to be assembled.

it enumerates all possible minimum rooted assembly subspaces that have $\gamma_y^{\max}$ as one of its longest rooted paths. In the next step, program p_4 picks the first of these possible minimum rooted assembly subspaces and encodes it in the form $\langle V\left({\Gamma^*}_y\right)\rangle$. Finally, program p_4 returns the output of program p_3 with $\mathbf{F}$ and $\langle V\left({\Gamma^*}_y\right)\rangle$ as inputs. The proof of Eq (10) follows from Eq (9) and basic inequalities in AIT. □

A sequence of edges in each of these longest (or shortest) paths constitutes a way to *parse* the object (e.g., a string), i.e., to decompose it into contiguous subsequences. Each distinct assembling path in ${\Gamma^*}_y$ corresponds to a distinct way to parse the object y. This idea will become clearer in the next Sections A.3.2 and A.3.3. It also constitutes the underlying method in compression schemes of LZ algorithms.

A.3.2 The assembly index defines a grammar of compression that is bounded by LZ compression and can at most approximate the Shannon Entropy rate

From here on, we investigate how well the assembly index can also produce an encoding whose size should be as small as possible in comparison to the size of the original object. In other words, we investigate how the assembly index performs as a *compression method*. Indeed, we show in the following sections that Assembly Theory is not only an encoding but also a *compression scheme*, in particular one that decomposes (i.e., parses) the object into contiguous building blocks, as is done by other methods like LZ compression algorithms and compressing grammars.

As presented in Section A.1, recall that the *assembly index* $c_\Gamma(x)$ of an object x is defined as the size[5] of the smallest acyclic quiver (directed multigraph) with edge-labellings from its own set of vertices[6], a quiver which is called the *assembly (sub)space*, denoted by (Γ, ϕ), or just Γ where the edge-labellings given by ϕ are not relevant. The "tree"-like structure of the assembly space maps the generation of an object from a set of basis objects in $B_{\mathcal{S}}$, which is a finite set and remains constant for any object assembled from combinations of the basis objects. Each vertex represents an object, and the directed edge relation specifies the *generation hierarchy* (i.e., the label of the oriented edge) of the object to be combined with the previous object (i.e., the source vertex to which the respective outgoing labelled edges are connected) in order to assemble a new object (i.e., the target vertex to which the respective incoming labelled edges are connected).

Now, given that these minimum rooted assembly subspaces are finite and acyclic, the number of objects that they can represent by an assembling process in a given assembly space is finite. Therefore, as demonstrated in the following Theorem 7, there will be a *Context-Free Grammar* (CFG) $G = (V_G, \Sigma_G, R, S)$ that generates each of these objects by combining terminal symbols from an alphabet. These terminal symbols correspond to the basis objects, where V_G is the alphabet of G, $\Sigma_G \subset V_G$ is the set of terminal symbols, and $R \subseteq (V_G \setminus \Sigma_G) \times V_G^*$ is the set of derivation rules, and $S \in (V_G \setminus \Sigma_G)$ is the start symbol.

Grammar-based compression algorithms have been an active area of research [11–14]. The problem of finding the smallest grammar has been studied for several decades and is known to be NP-Complete [12, 15]. Furthermore, the size of these compressing grammars is lower bounded by LZ encoding [13] (disregarding time complexity), which means that they converge to LZ in the optimal case for ergodic and stationary stochastic processes, and therefore are bounded by the noiseless compression limit [16].

It is straightforward to show the existence of such a CFG that encodes all the objects generated by an assembly space. In the following, we will investigate it further

[5] Except for the basis objects.

[6] And also by adding the property described in Item 6 in [5, Definition 11] of assembly spaces.

to demonstrate that for any ${\Gamma^*}_x$ (i.e., a minimal rooted assembly subspace) that generates a string x, there exists a grammar G that *encodes* this ${\Gamma^*}_x$, encoding not only the final object x to be assembled, but the whole assembling process rooted in the basis objects. In other words, there is at least one CFG that generates x and manifests a correspondence between: the basis objects of both the grammar (terminal symbols) and the assembly subspace; non-basis objects and vertices of ${\Gamma^*}_y$ with the sets of non-terminal symbols; and edges of ${\Gamma^*}_y$ with derivation rules of the grammar.

Theorem 7. *Let ${\Gamma^*}_x$ be an arbitrary minimum rooted assembly (sub)space from which the assembly index $c_\Gamma(x)$ is calculated. Then, there exists a CFG $G = (V_G, \Sigma_G, R, S)$ that generates x such that the equalities*

- $V\left({\Gamma^*}_x\right) \setminus \{x\} = V_G \setminus \Sigma_G$,
- $\Sigma_G = B_\Gamma \cup \{x\}$ *(correspondence between G terminal symbols and ${\Gamma^*}_x$ basis) and*
- $\left|E\left({\Gamma^*}_x\right)\right| + |B_\Gamma| = |R|$

hold, where B_Γ is the basis set of the assembly space (Γ, ϕ) from which x is one of its vertices/objects such that ${\Gamma^*}_x \subseteq \Gamma$. These equalities establish the correspondence between the terminal symbols of ${\Gamma^*}_x$ and the non-terminal symbols of G, the terminal symbols of G and the basis of ${\Gamma^*}_x$, and the edges of ${\Gamma^*}_x$ and the derivation rules of G, respectively.

Proof. Let ${\Gamma^*}_x$ generate x. In order to construct the corresponding CFG, we first define the set of terminal symbols in G as the set of basis objects in ${\Gamma^*}_x$ plus the final object x to be assembled, that is, $\Sigma_G = B_\Gamma \cup \{x\}$. Then, let $V_G \setminus \Sigma_G = V\left({\Gamma^*}_x\right) \setminus \{x\}$ be the set of non-terminal symbols, that is, we add to the set of non-terminal symbols every object in the assembly subspace ${\Gamma^*}_x$, except for the object to be assembled at the end. Now, we construct the set R of derivation rules of the CFG in the following way:

1. we start by adding $S \to x$ to the set of derivation rules for each basis object x that belongs to $V_G \setminus \Sigma_G$ and also for each basis object that belongs to Σ_G;
2. then, for each $e_i \in E\left({\Gamma^*}_x\right)$ such that $s_\Gamma(e_i) = x_i$ and $t_\Gamma(e_i) = y_i$, we construct a corresponding new rule, either $x_i \to x_i z_i$ or $x_i \to z_i x_i$, where $z_i \in V\left({\Gamma^*}_x\right)$ and $\phi\left(e_i\right) = z_i$, whichever concatenation of combinations builds the object/vertex y_i (i.e., either $y_i = x_i z_i$ or $y_i = z_i x_i$);
   (a) if $y_i = x_i z_i \neq x$ (or $y_i = z_i x_i \neq x$), then we add y_i to the set of non-terminal symbols $V_G \setminus \Sigma_G$.
   (b) otherwise, we add $y_i = x$ to the set of terminal symbols of G.

Notice that according to the above Step 1, there will be $2\,|B_\Gamma|$ rules in the form $S \to x$. Also notice that Step 2 is possible since each "assembly step" (represented by the respective edge) can only combine two objects: one object is the source vertex of the edge; and the other object is the vertex that is the label of this edge. From our construction of the set R, one will obtain that

$$\left|E\left({\Gamma^*}_x\right)\right| + |B_\Gamma| = |R| \ . \tag{11}$$

The rest of the proof that G generates x follows by induction over the number of vertices in $V\left({\Gamma^*}_x\right)$. If $\left|V\left({\Gamma^*}_x\right)\right| = 1$, then x must be a basis object in B_Γ. Hence, x is a terminal symbol in G, and thereby it directly follows that $S \to x$ generates x. Now, we assume that G generates any non-terminal symbol x_i for $i \leq n$ with a set R_n of derivation rules according to the definition and axioms of an assembly (sub)space [10], where each vertex x_i belongs to a rooted path (in ${\Gamma^*}_x$) whose terminal vertex is x. Let

$V\left(\Gamma^*{}_x\right) = \{x_1, \ldots, x_n, x_{n+1}, \ldots, x\}$ be the set of vertices so that $\Gamma^*{}_x$ generates x. By the inductive hypothesis, once there exists an edge e with $s_\Gamma(e) = x_i$ and $t_\Gamma(e) = x_j$ for some x_i and x_j with $i, j \leq n$, we will have it that there is the rule $x_i \to x_i z_i$ or $x_i \to z_i x_i$ in R_n, where either $x_j = x_i z_i$ or $x_j = z_i x_i$, respectively. By construction of $\Gamma^*{}_x$, we have it that there is a vertex $x'_i \in \{x_1, \ldots, x_n\}$ that is the source vertex of an edge $e' \in E\left(\Gamma^*{}_x\right)$ whose label is another $x'_j \in \{x_1, \ldots, x_n\}$ and the target vertex of e' is $t_\Gamma(e') = x_{n+1}$. Then, we add the rule $x'_i \to x'_i x'_j$ or $x'_i \to x'_j x'_i$ to R_n, where $x_{n+1} = x'_i x'_j$ or $x_{n+1} = x'_j x'_i$, respectively. Therefore, since by the inductive hypothesis both x'_i and x'_j can be generated by the set R_n of rules, we will have it that $R_n \cup \left\{x'_i \to x'_i x'_j \vee x'_i \to x'_j x'_i\right\}$ generates the object x_{n+1}. Finally, by the above construction of the set R of derivation rules, we will have it that if $x_{n+1} = x$, then x_{n+1} will be a terminal symbol and G stops exactly at the object x.

□

Notice that Theorem 7 holds for any arbitrary assembly (sub)space Γ, and not only for the minimal ones, demonstrating that any generative process modeled (or represented) by assembly spaces is equivalently modeled (or represented) by CFGs. In addition to the generative parity demonstrated in Theorem 7, the minimality of the assembly subspace (AS) will be employed in Section A.3.3 (see also Section A.3.1) so as to guarantee that one is able to encode (and compress) the objects. In this manner, one will obtain that any assembled object can be encoded into a CFG in Chomsky Normal Form (CNF) that captures the same information regarding how an object is constructed from other objects.

As presented in Section A.1.2, the assembly index is defined as the number of non-basis vertices in the minimum rooted AS. We have that the *size* $|G|$ of a CFG G (with its useless rules removed [13]) is defined as the number of derivation rules, or the number of nonterminals. When one has a minimum size of a generative context-free grammar that is able to generate the objects that another grammar G is capable of generating, we denote it by G^*. In order to avoid ambiguities, we denote by G_Γ the grammar as constructed in Theorem 7 for an arbitrary assembly (sub)space Γ.

We will show in Corollary 8 that there is a simple linear relationship between the assembly index $c_\Gamma(x)$ and the size of the minimal grammars, except for the size of the basis set B_Γ which remains constant for any object assembled upon combinations of basis objects.

Corollary 8. *Let x be a string/object assembled according to $\Gamma^*{}_x$. Then,*

- *the assembly index is equivalent to the non-basis symbols in $G_{\Gamma^*{}_x}$, i.e.,*

$$c_\Gamma(x) = \left|G_{\Gamma^*{}_x}\right| - |B_\Gamma| \ ; \tag{12}$$

- *the assembly index is lower bounded by the non-basis symbols in the minimum grammar that generates x, i.e.,*

$$c_\Gamma(x) \geq \left|G^*_{\Gamma^*{}_x}\right| - |B_\Gamma| \ ; \tag{13}$$

- *the number of factors (blocks or codewords) in the LZ encoded form of x, denoted by $LZ(x)$, is upper bounded by the assembly index (except for the basis set B_Γ that is finite and constant), i.e.,*

$$LZ(x) \leq c_\Gamma(x) + |B_\Gamma| \ . \tag{14}$$

Proof. The first equation is by construction of G in Theorem 7. The second one follows directly from the first one. The third one is a direct consequence of the second one and [13, Theorem 1]. □

Corollary 8 proves that the assembly index for strings is equivalent to the compression size of a compressing grammar, demonstrating that the notion of the minimum "amount of assembling steps" (that the minimum rooted assembly subspace defines) is exactly the same as a (minimal) CFG. For each of the grammars constructed in Theorem 7, the final object x is unique, and for each distinct assembling path leading to x, there is a (unique) corresponding sequence of rules in R that also results in the object x.

In the optimal case for stochastic sources, such a size can only converge towards a traditional LZ compression scheme [17–19], such as LZ77 and LZ78, and thus its compression rate capabilities are bounded by Shannon Entropy by the (noiseless) Source Coding Theorem [16, 20]. It is important to note that the strict inequality in Equation (14) arises when the corresponding grammar, thus the assembly space, is *not* as optimal at identifying the "building blocks" of x (given the same alphabet of basis objects) as an LZ factorization is. For compressing grammars in particular, several implementation-specific entropy bounds were found in [14].

Now, for higher-dimensional objects, if the construction is recursive, it is possible to find an one-dimensional rule that encodes the object's assembly. For example, for a two-dimensional object, we can construct a grid of fixed size around the object and incorporate the positional information into the grammar's rules while maintaining a linear relationship between the assembly index and the size of the grammar. The rule $X \to ApB$ can define how object B is added to object A at position p, where p corresponds to a square in the grid, to generate object X.

A.3.3 The assembly index is a version of LZ compression

The most popular examples used by the authors of Assembly Theory (AT) in their papers as illustrations of the way their algorithms work, such as ABRACADABRA and BANANA, are traditionally used to teach LZ77, LZ78, or LZW compression in Computer Science courses at university level. Dictionary trees have been used for pedagogic purposes in computer science for decades. In addition, beyond a notation '$c(x)$' for the formalisation of the assembly indexes (and assembly spaces) [10] that resembles that of the number of phrases (or factors) in the parsings of LZ78 [16], they also share underlying key ideas in the relationship between these parsings and the probability of the phrases into which the objects are decomposed. One of the central motivations for the assembly index as a complexity measure is that the probability of assembling paths that leads to an object should decrease as the respective assembly index (or, in the case of molecules, MA) of the object increases [7], unless there are environmental constraints or an extrinsic agent responsible for increasing the frequency of occurrence of high-assembly-index objects. Similarly, this appears as the key idea in the proof that LZ78 achieves optimal compression [16] for stationary and ergodic stochastic processes: sequences with a larger number of distinct phrases to which the pointers necessarily recur less often would have lower probability, and therefore they are less compressible; while sequences with fewer distinct phrases corresponds to a higher probability, and therefore they can be compressed more efficiently.

At first glance, these above aspects suggest some similarity between AT and compression algorithms, particularly those of a statistical nature based on recursion to previous states, copy counting, and the re-usage of patterns and repetitions. Indeed, such an interdisciplinary approach between physics, chemistry, and computer science is very beneficial and fruitful for science in general, specially in the field of complex systems science.

In Section A.3, we have demonstrated that either the set of vertices of a minimal rooted assembly subspace ${\Gamma^*}_y$ or *at least one* of the longest (or shortest) rooted paths in this ${\Gamma^*}_y$ is sufficient to encode the object y. These results employ general algorithmic

methods so that they are independent of the choice of programming language and encoding-decoding schemes. In Theorem 7, we have demonstrated that every minimum rooted assembly (sub)space (AS) corresponds to *at least one* context-free grammar (CFG), which reveals a way one can map an AS, e.g. the minimum rooted assembly subspace ${\Gamma^*}_y$ that generates the object y, onto the minimum set of rules of a CFG which compresses the object. These results together also indicate the existence of some LZ-based method to encode both the object and its assembling process.

Indeed, the proof of the following Theorem 9 constructs such a LZ compression scheme defined by the assembly index calculation method (to which we refer as 'LZAS'). In contrast to the claims of AT's authors, we have shown in the present article that the calculation of the assembly index from the minimum rooted AS in fact is a *compression scheme* belonging to the *LZ family* of compression algorithms, rather than merely being similar. The present study also theoretically establishes and extends the limitations previously investigated in [8].

For traditional LZ schemes, the encoded/compressed form of a string comprises a sequence of encoded tuples (factors, phrases or codewords) of finite length [16, 21]. Typically, each one of these tuples contains at least one pointer (e.g., in the case of LZ78 and LZW, called indexes) that refers to another tuple or to an entry in a dictionary. Also, it can contain information about some elements of the basic alphabet (e.g., in the case of LZ78 and LZW it contains the codeword for only one symbol to be concatenated at the end of the element the pointer is referring to). It is a well-known trivial property in data compression and coding theory that one can find examples of objects whose parsing according to the LZ78 differs from the parsing according to the LZ77, or that the one according to the LZMA differs from LZW, and so on. AT's authors later investigated this property in [22] by presenting individual examples in which the parsing according to the assembly index calculation method differs from LZW's. As earlier demonstrated in the present article—results which seem to be overlooked in [22]—, this is in accordance with the theoretical expectation for a compression scheme member of the LZ family. The argument that the assembly index calculation method is not LZ compression because its parsings differ from those of LZW is as pertinent as the argument that LZW is not LZ compression because its parsings differ from those of LZ77.

The LZ family tree of compression schemes includes many distinct schemes, including LZ77, LZ78, LZW, LZSS, and LZMA [21]; and as we demonstrate in Theorem 9 below, it also includes the assembly index calculation method. All of these compression methods are defined by schemes that resort to a (static or dynamic) dictionary containing tokens, indexes, pointers, or basic symbols from which the decoder or decompressor can retrieve the original raw data from the received encoded/compressed form according to the respective LZ scheme [21]. Theorem 9 demonstrates that the minimum rooted assembly (sub)space from which the assembly index is calculated is a compression scheme (namely, 'LZAS') that belongs to the LZ family of compression algorithms.

On the one hand, the pointers in traditional LZ compression schemes, such as LZ77, LZ78 or LZW, refer to encoded tuples that have occurred before as recorded by the associated dictionary. On the other hand, for Assembly Theory, the pointers refer to other tuples that might not have occurred before in a particular assembling path, but are part of the minimum rooted assembly subspace ${\Gamma^*}_x$, that is, part of a "tree"-like structure that needs to be encoded in the dictionary, which is only built or queried during the encoding/decoding process.

In practice, the major characteristic that distinguishes assembly index compression from other traditional LZ schemes (like LZ77/78 or LZW), is that: in the latter case, one would parse the string into factors, that is, contiguous smaller strings (or

codewords) whose resulting unidimensional concatenation directly represents the string; while in the former case, one would parse the object into a non-unidimensional structure where each block is "concatenated" according to a directed acyclic multigraph with edge labels drawn from the set of vertices. For a minimum rooted assembly (sub)space there may be more than one rooted path terminating at the final object to be assembled. Each of these paths corresponds to a distinct way to parse (i.e., decompose) the object.

As we will demonstrate in Theorem 9 that assembly index calculation method belongs to a LZ scheme, the sequence of edges that constitutes an assembling path (from the basis set B_Γ to the terminal object/vertex in a minimum rooted assembly subspace ${\Gamma^*}_y$) can be converted into a sequence of respective encoded tuples that contains pointers, codewords for other objects, etc. One can think of these assembling paths being, for example, the sequence defined by either one of the longest rooted paths $\gamma_x^{\max}$ or one of the shortest rooted paths $\gamma_x^{\min}$ in the minimum rooted assembly subspace ${\Gamma^*}_x$. In particular, the *shortest path* is often cited by AT's authors as the one that translates or captures the idea of complexity that the assembly index is aiming at quantifying [5–7, 9, 10].

Indeed, as demonstrated in the upcoming Theorem 9, the length of the shortest paths is the same (except for the four initial blocks of encoded data, i.e., the 0-th initial tuples) as the number of factors (or the number of tuples of pointers) according to the parsing in the encoded form (see Eq (16)) of 'LZAS'. The latter refers to the LZ scheme defined by the minimum rooted AS from which the assembly index is calculated. Thus, as shown in Theorem 9, 'LZAS' establishes a way to compress such a non-linear structure (in particular, a directed acyclic multigraph) and non-deterministic parsings/decompositions back into a linear sequence of codewords. The compressibility of this sequence is always lower-bounded by the length of the shortest assembling paths. We demonstrate that the minimality of ${\Gamma^*}_x$ in fact guarantees that such a scheme (where pointers may *not* refer to the unidimensional past subsequences) allows unique decoding, despite the non-unidimensional structure of assembly spaces in which more than one path may lead to the same object. In the same manner as the dictionary of LZW that has to be initialized before the sequence of pointers is encoded/decoded, the 0-th initial tuples encode the 'LZAS' dictionary to be employed in the encoding/decoding of the forthcoming sequence of tuples of pointers. This in turn demonstrates that the CFG compression scheme equivalence in Section A.3.2 can be indeed understood as an 'LZ-grammar' compression: the set R of derivation rules of an AS plays the role of a dictionary from which the decoding scheme can decompress the sequence in Eq. (16) into the object; and $\langle t \rangle$, $\left\langle \left| \gamma_y^{\max} \right| \right\rangle$, and $\langle v_B \rangle$ constitute an encoding of the amount of information to describe the minimal dictionary.

Theorem 9 (LZ encoding from the assembly index calculation method)**.** *Let $\mathcal{S}$ be enumerable by an algorithm. Let* **F** *be an arbitrary formal theory that contains Assembly Theory, including all the decidable procedures of the chosen method for calculating the assembly index of an object for a nested subspace of $\mathcal{S}$, and the program that decides whether or not the criteria for building the assembly spaces are met. Let $\Gamma \subset \mathcal{S}$ with $y \in V(\Gamma)$, $\gamma_y^{\min}$, and $\gamma_y^{\max}$ be arbitrary. Then, one has it that the sequence $S_{AT}(y)$ composed of $\left| \gamma_y^{\min} \right|$ encoded tuples in the form $\langle c, i_t, v_l \rangle$ (except for the 0-th initial tuples which are $\langle t \rangle$, $\left\langle \left| \gamma_y^{\max} \right| \right\rangle$, $\langle B_\Gamma \rangle$, and $\langle v_B \rangle$, respectively, where $v_B \in B_\Gamma$ and $t \in \mathbb{P}(\mathcal{T})$)*

encodes the assembling process of the object y such that

$$\begin{aligned} \left|\gamma_y^{\min}\right| \pm \mathbf{O}(1) \leq \\ |S_{AT}(y)| \leq \\ \mathbf{O}\left(2^{3\left|\gamma_y^{\max}\right|+6} + |B_\Gamma| \log(|B_\Gamma|)\right) + \mathbf{O}\left(\log\left(\left|\gamma_y^{\max}\right|\right)\right) + \mathbf{O}\left((|B_\Gamma|+1)\log(|B_\Gamma|)\right) + \\ \left|\gamma_y^{\min}\right| \left(\mathbf{O}\left(3\left|\gamma_y^{\max}\right|+6\right) + \log\left(\left|\gamma_y^{\min}\right|\right)\right) \leq \\ \mathbf{O}\left(2^{3\,c_\Gamma(y)+6} + |B_\Gamma| \log(|B_\Gamma|)\right) + \mathbf{O}\left(\log\left(c_\Gamma(y)\right)\right) + \mathbf{O}\left((|B_\Gamma|+1)\log(|B_\Gamma|)\right) + \\ c_\Gamma(y)\left(\mathbf{O}\left(3\,c_\Gamma(y)+6\right) + \log\left(c_\Gamma(y)\right)\right) \end{aligned} \tag{15}$$

holds, where:

- $\left|\gamma_y^{\min}\right|$ *is the length of the rooted path* $\gamma_y^{\min}$*, which is one of the shortest rooted paths with* y *as terminal vertex that belongs to the minimum rooted assembly subspace* ${\Gamma^*}_y$*;*
- $\left|\gamma_y^{\max}\right|$ *is the length of the rooted path* $\gamma_y^{\max}$*, which is one of the longest rooted paths with* y *as terminal vertex that belongs to the minimum rooted assembly subspace* ${\Gamma^*}_y$*;*
- $\left\langle\left|\gamma_y^{\max}\right|\right\rangle$ *is an encoding of the number* $\left|\gamma_y^{\max}\right|$*;*
- $\langle B_\Gamma\rangle$ *is an encoding of the basis set* B_Γ*;*
- $\langle v_B\rangle$ *is an encoding of the vertex* $v \in B_\Gamma$ *which is the initial object/vertex of* $\gamma_y^{\min}$*;*
- c *is the counter of the tuple in the sequence* $S_{AT}(y)$ *(i.e., the natural number that indicates the position of the tuple) with* $1 \leq c \leq \left|\gamma_y^{\min}\right|$*;*
- $\langle t\rangle$ *is an encoding of the* t*-th member of* $\mathbb{P}(\mathcal{T})$*, where* $\mathbb{P}(\mathcal{T})$ *is the powerset of possible edges of the minimum directed acyclic multigraph* $\mathcal{T}$ *into which any possible minimum rooted assembly subspace (whose longest rooted paths have length* $\left|\gamma_y^{\max}\right|$*) can be embedded, and all the most distant vertices from the terminal vertex (i.e.,* y*) can only be the vertices in* B_Γ*;*
- i_t *is the index/pointer that refers to the corresponding edge in* $\mathcal{T}$*;*
- v_l *is another index/pointer that refers to the edge* i'_t *whose source vertex* $s(i'_t) = v \in V(\Gamma)$ *is the vertex that is the label of the edge* i_t *when* Γ *is embedded into* $\mathcal{T}$*;*
- $|S_{AT}(y)|$ *denotes the length (in bits) of the encoded sequence* $S_{AT}(y)$*;*
- $c_\Gamma(y)$ *is the assembly index of the object* y *in the assembly space* Γ *(or* $\mathcal{S}$*).*

The encoding-decoding scheme for Theorem 9. Let p_5 be a program that receives a sequence $S_{AT}(y)$ in the form

$$\langle t\rangle \left\langle\left|\gamma_y^{\max}\right|\right\rangle \langle B_\Gamma\rangle \langle v_B\rangle \langle 1, i_{t1}, v_{l1}\rangle \langle 2, i_{t2}, v_{l2}\rangle \langle 3, i_{t3}, v_{l3}\rangle \ldots \left\langle\left|\gamma_y^{\min}\right|, i_{t_{\left|\gamma_y^{\min}\right|}}, v_{l_{\left|\gamma_y^{\min}\right|}}\right\rangle \tag{16}$$

as input. Then, by employing $\mathbf{F}$, it constructs the directed acyclic multigraph (DAmG) $\mathcal{T}$ of height $\left|\gamma_y^{\max}\right|$, and enumerates all the possible minimum rooted assembly subspaces contained in $\mathcal{T}$ via the following procedure: it starts from those that are isomorphic to an arbitrarily fixed $\gamma_y^{\max}$ with any possible combination of initial

vertex/object in B_Γ; and then in a dovetailing procedure, it indexes first those subspaces that differ (from this $\gamma_y^{\max}$) by the least amount of additional edges. Then, program p_5 chooses the t-th first subspace Γ' that appears in this enumeration. In the next step, program p_5 reads the rest of the sequence $S_{AT}(y)$, extracting from the encoded tuples the sequence $I = \left(i_{t1}, \ldots, i_{t|\gamma_y^{\min}|}, v_{l1}, \ldots, v_{l|\gamma_y^{\min}|}\right)$ of indexes of edges that are contained in this t-th first Γ' such that it constitutes a shortest rooted path that starts at vertex v_B. Finally, it uses this shortest rooted path to assemble the final object y' accordingly. This describes the *decoding* procedure from $S_{AT}(y)$. In order to construct a program p_5' that *encodes* a minimum rooted assembly subspace ${\Gamma^*}_y$ into a $S_{AT}(y)$, one just needs to note that the codewords for $\langle|\gamma_y^{\max}|\rangle$, $\langle B_\Gamma\rangle$, and $\langle y\rangle$ are unique and always retrievable by an algorithm given ${\Gamma^*}_y$ as input. From these, program p_5' can use the same procedures that p_5 ran in order to construct $\mathcal{T}$. Program p_5' then employs the same procedures that p_5 ran in order to enumerate the minimum rooted assembly subspaces embedded into $\mathcal{T}$ so as to find the index t that corresponds to ${\Gamma^*}_y$ in this enumeration. Next, it enumerates all the possible shortest rooted paths in this t-th minimum rooted assembly subspace. Then, it extracts v_B and I from $\mathcal{T}$ by matching those that correspond to the edges and their respective labels in the first shortest rooted path $\gamma_y^{\min}$ found in the previous enumeration that terminates at the object y. Finally, the program returns the sequence $S_{AT}(y)$ as output. □

Proof of Theorem 9. With the purpose of demonstrating that $\Gamma' = {\Gamma^*}_y$, first note that there is only a finite number of rooted assembly spaces whose most distant vertices (from the vertex/object y) belong to B_Γ (which is also finite and fixed given Γ) and are separated by $|\gamma_y^{\max}|$ hops. Therefore, since $\langle|\gamma_y^{\max}|\rangle$ and $\langle B_\Gamma\rangle$ are invariant for each Γ in $\mathcal{T}$ (including ${\Gamma^*}_y$) and the enumeration procedure is the same in both the encoding and the decoding phase, there is a unique index t that corresponds to $\Gamma' = {\Gamma^*}_y$. In order to achieve the proof of Eq (15), one can exploit the minimality of ${\Gamma^*}_y$ and $\gamma_y^{\min}$ and the maximality of $\gamma_y^{\max}$. From any rooted assembly space one of whose longest rooted paths is $\gamma_y^{\max}$, in order to construct $\mathcal{T}$, an algorithm can minimise the number of sufficient and necessary new vertices that need to be added to $\gamma_y^{\max}$ using the following procedure: assign an index $j \in \mathbb{N}$ to each of the edges in $\gamma_y^{\max}$ starting from the closest and moving to the farthest ones from the initial vertex, which is the object in the basis set B_Γ, such that $1 \leq j \leq |\gamma_y^{\max}|$; for the label of the edge $j = 1$ in $\gamma_y^{\max}$, one only requires at most one extra vertex, and this vertex can only belong to B_Γ because otherwise, since there would need to be an additional edge connecting this new vertex to the target vertex of edge $j = 1$, it would contradict the maximality of $\gamma_y^{\max}$; the same analogously holds for each arbitrary $j + 1 \leq |\gamma_y^{\max}|$, so that labelling this edge $j + 1$ with a vertex only requires at most one extra new vertex, and this vertex can only be at level $\leq j$, i.e., it can only belong to the set of vertices that are separated from the terminal vertex y by at least $|\gamma_y^{\max}| - j + 1$ hops (otherwise, it would again contradict the maximality of $\gamma_y^{\max}$); and for each new vertex created at level j one would only need at most 2 new vertices at level $\leq j - 1$, and so on. Thus, by induction and the minimality of ${\Gamma^*}_y$, one will have it that

$$\sum_{i=0}^{|\gamma_y^{\max}|} 2^i = 2^{|\gamma_y^{\max}|+2} \tag{17}$$

is an upper bound for the maximum number of distinct vertices that a minimum rooted assembly subspace (whose $\gamma_y^{\max}$ is one of its longest rooted paths) can have. Following a similar inductive argument upon the maximality of $\gamma_y^{\max}$, one concludes that

$$2^{3\,|\gamma_y^{\max}|+6} \tag{18}$$

is an upper bound for the maximum number of distinct (multi)edges that a minimum rooted assembly subspace can have. Hence, one only needs at most

$$\mathbf{O}\left(\log\left(2^{3\left|\gamma_y^{\max}\right|+6}\right)\right) = \mathbf{O}\left(3\left|\gamma_y^{\max}\right|+6\right) \tag{19}$$

bits to encode either each i_t or each v_l. (Notice that there may be tighter upper bounds than those in Equations (17) and (18), but for the purposes of Theorem 9 they suffice). Therefore, we have it that ${\Gamma^*}_y$ can be embedded into $\mathcal{T}$, and as a consequence there will be a $t \in \mathbb{N}$, where

$$1 \leq t \leq \left(2^{2^{3\left|\gamma_y^{\max}\right|+6}}\right)\ |B_\Gamma|! \tag{20}$$

holds, such that t corresponds to this exact structure given by ${\Gamma^*}_y$ when program p_5 starts to enumerate all possible minimum assembly subspaces embedded into $\mathcal{T}$. Then, one needs at most

$$\mathbf{O}\left(\log\left(\left(2^{2^{3\left|\gamma_y^{\max}\right|+6}}\right)\ |B_\Gamma|!\right)\right) \leq \mathbf{O}\left(2^{3\left|\gamma_y^{\max}\right|+6} + |B_\Gamma|\log\left(|B_\Gamma|\right)\right) \tag{21}$$

bits to encode this t. By knowing the exact DAmG structure of ${\Gamma^*}_y$, and to which edge each basis vertex is connected, an algorithm can promptly employ $\mathbf{F}$ to reconstruct both every object assembled across this DAmG structure and every rooted path that terminates at each of these objects. Now, from the minimality of $\gamma_y^{\min}$ and from the definition of assembly space and the assembly index, we have it that $\left|\gamma_y^{\min}\right| \leq \left|\gamma_y^{\max}\right| \leq c_\Gamma(y)$. Therefore, from basic inequalities known from the theory of algorithmic complexity, when applied to the best way to encode each term in the sequence in Eq (16), one obtains the proof of Eq (15). □

In relation to the claims made by the authors of Assembly Theory [5–7, 9, 10], Theorem 9 reveals that Assembly Theory's fundamental intuition to classify the notion of complexity of assembled objects is not only that of a compression scheme, but specifically an LZ scheme. It is a member of the LZ family of compression methods. The non-unidimensional "tree"-like structure of the minimum rooted assembly subspaces is one of the ways to define a scheme—constituted by a dictionary and a sequence of pointers—that encodes the assembling process itself back into a linear sequence of codewords. Shorter minimum paths lead to more compressed forms of the assembling process of the objects. That is, 'LZAS' (which is the LZ encoding-decoding scheme behind Eq (16)) translates the notion of the shortest assembling path playing a role in how much simpler the object can get, i.e., as we have demonstrated, how much more compressible the object is.

Finding a minimal rooted assembly subspace for an object is a computationally expensive problem, as demonstrated in Theorem 9, in Section A.3.2, and acknowledged by AT's authors [5–7, 9, 10]. (In addition, as we will discuss in Section A.3.4, the exponential growth in 'LZAS' that our theorem demonstrates is, in turn, consonant with the assembly number [6] of an ensemble). Theorem 9 demonstrates that "simpler" minimum rooted assembly subspaces lead to more compressibility. This notion of simplicity encompasses both how much the assembly space structure differs from a single-thread (or linear) space and how many more distinct assembling paths can lead to the same object. If the search for the minimal assembly subspaces needs to cover a wider space of possible assembling paths in order to calculate the assembly index, then this process may turn out to be computationally expensive. To tackle this intractability, AT employed some approximation methods to the calculation of the assembly index, such as the split-branch version [5, 10], thus taking advantage of a narrower search by constraining the potential assembly subspaces.

Indeed, the proof of Theorem 9 clarifies the fact that any (whether more computationally efficient or not) approximation method, e.g. the split-branch version [5, 10], not only is a LZ scheme but also a variation of 'LZAS' that may improve on computational resources costs and/or compression rate: the more efficiently the approximation method gets closer to the (actual, but intractable in the general case) value of assembly index, the more the corresponding variation of (the actual) 'LZAS' improves on compression rate and/or the usage of computational resources to achieve such a compression rate.

For example, the reader is invited to note that due to the fact that $\gamma_y^{\max} = \gamma_y^{\min}$ would hold, a linear minimum rooted assembly subspace ${\Gamma^*}_y$ (such as in [7, Figure 1(a)], [5, Figure 8], or [22, Figure 3]) results in a much tighter upper bound in Eq (15). This is because the exponential dependence on the assembly index to encode all possible DAG-like structures would drop: that is, $1 \leq t \leq \mathbf{O}\left(\left|\gamma_y^{\max}\right|^2 |B_\Gamma|\right)$ will hold instead of Eq (20), which in turn will be subsumed in the rightmost term of Eq (15) where one already has a quadratic dependence on $\left|\gamma_y^{\max}\right|$ or $c_\Gamma(y)$. Once a less linear (or more DAG-like) assembling structure unfolds, as $\gamma_y^{\max}$ begins to differ more from $\gamma_y^{\min}$, the encoding or decoding algorithms in Theorem 9 will have to enumerate more distinct possible minimum rooted assembly spaces, and as a consequence it will increase the upper bound for $|S_{AT}(y)|$, which also renders the encoding and decoding process slower. Conversely, the more straightforward the DAmG structure for which one is calculating or approximating the assembly index, the narrower the search for minimum spaces candidates; and as consequence the more compressible and more computationally efficient the compression scheme will be.

The authors in [5, 10] pointed out that if one tries to apply the assembly index calculation method to compress a string, it would result in a computationally expensive algorithm. For example, in the abstract of Marshall et al. [7] it is stated that their method is the first "experimentally tractable measure of molecular complexity", which is in principle an inconsistent statement with regards to the very assembly index method. Therefore, the very tractability of the purportedly measure is only sound in the case of *approximations* that are distorted [8] in the first place. Because the assembly index calculation method is a LZ compression scheme, our results in this section demonstrate that such an intractability not only arises in potential applications of AT to coding and compression, but also is the very intractability which *in fact is intrinsic* to AT, occurring whenever one tries to calculate the assembly index for an assembly space—a process for which AT had to employ approximation techniques, such as the split-branch assembly space.

Together with the height of the grammars in Theorem 7, the interdependence between compressibility and computational efficiency in the proof of Theorem 9 also captures the notion of "depth" that AT intends to attain but that in fact pertains, as we have demonstrated, to a compression scheme and a generative grammar. The more complex the DAG-like structure of ${\Gamma^*}_y$ is, the more distinct DAG-like structures are possible, and therefore the closer the upper bound becomes to the exponential dependence. Hence, Eq (15) would render the object (or assembling process) less compressible and more computationally demanding, which demonstrates the claims underlying Assembly Theory that the authors in [5–7, 9, 10] originally intended to demonstrate with the assembly index but that in fact we demonstrated with a compression scheme. These results show that the notion of how complex an assembling process needs to be in order to construct an object is indeed equivalent to how much less compressible (by 'LZAS') and more computationally demanding (according to this encoding-decoding LZ scheme) this process is.

Thus, Theorem 9 single-handedly demonstrates that:

- the assembly index calculation method is a *LZ scheme*;
- and that the *intractability* of the calculation of the assembly index (or, equivalently, the intractability of the encoding-decoding procedure) *is intrinsic* to the notion of complexity that the assembly index intended to measure in the first place.

Further, as in Section A.3.4, one can demonstrate that any measure defined in terms of the assembly index—so that it increases when the assembly index(es) also increases—, such as the assembly number [6], also defines an encoding and a compression method. The higher the assembly number of an ensemble, the more the ensemble is compressible and the other way around, in direct and equal proportion.

A.3.4 The assembly number defines an encoding and a compression scheme

As proposed in [6], the assembly number $A\#(X) = \sum_{i=1}^{N} e^{a_i} \left(\frac{n_i - 1}{N_T}\right)$ is intended to measure the amount of selection necessary to produce the ensemble X, i.e., the presence of constraints or biases in the underlying generative processes (e.g., the environment in which the objects were assembled) that set the conditions for the appearance of the objects. A higher value of $A\#(X)$ would mean that more "selective forces" were put into play as environmental constraints or biases in order to allow or generate a higher concentration of high-assembly-index elements. Otherwise, these high-assembly-index objects would not occur as often in the ensemble. This is because in stochastically random scenarios the probability would rapidly decrease as the assembly index increases.

The value of $A\#$ is calculated from the parameters N_T, N, n_i, a_i, which are retrievable by an algorithm from a given ensemble X whose number of copies of each unique object is greater than or equal to 1. The constant N_T (i.e., the total number of objects in the ensemble X) allows one to compare the assembly number between distinct ensembles whose sizes differ.

Notice that each term e^{a_i} plays the role of weight for the respective ratio/proportion $\frac{n_i - 1}{N_T} \leq 1$. In this manner, one can, for example, normalise again for ensembles whose overall summation of assembly indexes remains the same, hence only differing by the concentration/distribution of the quantity of objects with higher assembly index. As stated in [5–7, 9], grasping this distinction in fact is the main goal that Assembly Theory aims at with its methods, so as to measure the presence of constraints or biases that turn otherwise unlikely objects into objects that occur more often. However, as we will demonstrate below, AT is a simplistic approximation (whose methods and theory are fully *subsumed into algorithmic information theory*) to algorithmic complexity: it is a defective approximation that *lacks group-control* investigation on both empirical applications and theory, thus failing to avoid false positives and to take into account top-down causation.

First, notice that for any two ensembles X and X' with the same total number N_T of objects and whose overall summation $\sum_{i=1}^{N} e^{a_i}$ of assembly indexes is also the same, the ensemble with the higher concentration of high-assembly-index objects (with respect to the concentration of these objects in the other ensemble) should score higher on the measure that assembly theory aims to develop. This prevents a (n equal-size) comparison between a biased sampling and another sampling in which the former was taken under distinct conditions in the latter. In this case, without normalising by the summation $\sum_{i=1}^{N} e^{a_i}$, one could have the first one scoring a higher assembly number than the second one, but only because of the set of unique objects, and not because of the distribution of higher-assembly-index objects with respect to lower-assembly-index objects, which was what Assembly Theory intended in the first place. For example, to

tackle this issue, the value given by the measure

$$A'\#\left(X\right)=\frac{\sum\limits_{i=1}^{N}e^{a_i}\left(\frac{n_i-1}{N_T}\right)}{\sum\limits_{\substack{x_j\in\mathbf{x}\\ \sum\limits_{i=1}^{N}e^{a_i}=\sum\limits_{i=1}^{N_j}e^{a_{ij}}}}\left(\sum\limits_{i=1}^{N_j}e^{a_{ij}}\left(\frac{n_{ij}-1}{N_{T_j}}\right)\right)} \tag{22}$$

improves on the originally proposed measure $A\#\left(X\right)$ in [6].

Thus, *future research* in Assembly Theory is necessary to compare the performance and statistical relevance of $A'\#\left(X\right)$ and $A\#\left(X\right)$ beyond a restrict or small set of organic and inorganic compounds. A more pervasive investigation on those group-controlled versions of the assembly number may help reveal the presence of any theoretical bias or arbitrariness in AT, such as the presence of false positives due to abiotic mimicry [23, 24] with respect to the assembly index—which was previously demonstrated in [8], investigated in [25] for a particular class of abiotic compounds, and we further discuss below—, an even better control group for excluding sampling condition biases in experiments could be achieved by only comparing ensembles with not only equal size and equal total summation of assembly indexes but also the same set of unique objects.

In the general finite case, if the set of possible objects is finite and the maximum size of the ensembles investigated is also finite, then there is a finite set $\mathbf{X}$ of possible unique ensembles. In this scenario, one can always keep effecting further renormalisations upon Eq (22) to account for any possible ensemble in $\mathbf{X}$, and hence addressing a wider range of control groups in their experiments. (Notice that the forthcoming results in Theorem 10 with regards to assembly number hold analogously for any further normalisation than the one already given by Eq (22).)

In the general stochastic case, not only in the *finite* case but also in the *infinite* case, for any other measure $A''\#\left(\cdot\right)$ that one can possibly devise such that this measure is monotonically dependent[7] on $A\#\left(\cdot\right)$ (e.g., by applying any form of normalisation), one can directly apply the Krafts inequalities and the source coding theorem [16], should the ensembles be indeed sampled accordingly and the number of samples sufficiently large: one constructs an optimal code that statistically compresses each ensemble X in sequences whose length is $C'_A\left(X\right)$ such that

$$-\log\left(A''\#\left(X\right)\right)\leq C'_A\left(X\right)\leq-\log\left(A''\#\left(X\right)\right)+\mathbf{O}(1) \tag{23}$$

is expected to hold on average. As proposed by Assembly Theory, notice from Eq (23) that ensembles with higher assembly numbers are expected to be more compressible, and therefore they diverge more from those ensembles that are more statistically random. This *demonstrates* that in the case of pure *stochastic processes*, the assembly number is either a *suboptimal* or *an optimal* compression method, therefore adding *no* advantage in comparison to (Shannon) entropy-based methods.

If the set of possible ensembles can be arbitrarily large and the ensembles are *not* generated by stationary and ergodic sources—for example, in this case the minimum expected codeword length per symbol *may not converge to the entropy rate*—, then finding the optimal code is not as straightforward as finding the code that generated Eq (23). To tackle this generalised scenario, algorithmic complexity improves on any entropy-based coding methods, or on any pure statistical compression method. By using algorithmic information theory (AIT), algorithmic complexity being one of its indexes, one obtains the universally optimal encoding, and hence one demonstrates in

[7] That is, if $A\#\left(X_i\right)\leq A\#\left(X_j\right)$, then $A''\#\left(X_i\right)\leq A''\#\left(X_j\right)$.

Theorem 10 and Corollary 11 that *the assembly number is an approximation or estimation of the compressibility of the ensemble*[8]. That is, the assembly number calculation method is subsumed into the universal coding of AIT, demonstrating that the assembly number, in fact, attempts to capture the idea of compressiblity of the ensemble—not to be conflated with the object— although only as an approximation that can, in general, be arbitrarily distorted (see Section A.2.1).

Theorem 10. *Let $\mathcal{S}$ and $\mathbf{X}$ be enumerable by an algorithm. Let $A''\#(\cdot)$ be an arbitrary measure (or semimeasure) that can be effectively calculated by an algorithm from the assembly number $A\#(\cdot)$. Let $\mathbf{F}'$ be an arbitrary formal theory that contains Assembly Theory, including all the decidable procedures of the chosen method for calculating the assembly number $A\#$, the measure $A''\#(\cdot)$ for any subspace of $\mathcal{S}$ and ensemble in $\mathbf{X}$, and the program that decides whether or not the criteria for building the assembly spaces or ensembles of objects are met. Let $X \in \mathbf{X}$ be an arbitrary ensemble. Then, one can encode/compress the ensemble X in $C_A(X)$ bits such that*

$$-\log_2\left(A''\#(X)\right) \leq C_A(X) \leq -\log_2\left(A''\#(X)\right) + \mathbf{O}(1) \tag{24}$$

and

$$\mathbf{K}(X) \leq C_A(X) + \mathbf{O}(1) \tag{25}$$

hold, where $C_A(X)$ denotes the size of the codeword/program for the ensemble X.

Proof. By hypothesis we have it that there is an algorithm that can always calculate the value of $A''\#(\cdot)$ for any X. Hence, there will be another algorithm that can always calculate

$$l_X \coloneqq \lceil -\log_2\left(A''\#(X)\right)\rceil \tag{26}$$

from any ensemble X. Then, one directly obtains that the sequence

$$\left((l_{X_1}, X_1), \ldots, (l_{X_i}, X_i), \ldots\right) \tag{27}$$

is computable by an algorithm. In addition, due to the fact that $A''\#(\cdot)$ is a (semi)measure by hypothesis, one has it that

$$\sum_{X_i \in \mathbf{X}} \frac{1}{2^{l_{X_i}}} \leq 1 \ . \tag{28}$$

From AIT [4], one knows that if an arbitrary sequence $\left((d_1, \tau_1), \ldots, (d_i, \tau_i), \ldots\right)$ of pairs is computable and $\sum_i 2^{-d_i} \leq 1$, then one can encode each τ_i in a codeword of length d_i. Additionally, one also has it that

$$\mathbf{K}(\tau_i) \leq d_i + \mathbf{O}(1) \ . \tag{29}$$

Therefore, from basic properties and inequalities in AIT, we will have it that there is an algorithm that can compress X in a program of length $C_A(X)$ such that Equations (24) and (25) hold. □

Now, from Theorem 10, one can employ a strictly monotone dependence on the assembly number to demonstrate that a higher assembly number indeed implies greater compressibility of the ensemble.

[8] Note that the compressibility of the ensemble should not be confused with compressibility of the object as well as the assembly number of an ensemble should not be confused with assembly index of an object.

Corollary 11. *Let the conditions for Theorem 10 be met such that $A''\#(\cdot)$ is also* strictly increasing *on the assembly number $A\#(\cdot)$. Then, there is k_0 such that, if $X, X' \in \mathbf{X}$ are ensembles with $A\#(X) + k_0 \leq A\#(X')$, one has it that*

$$C_A(X') < C_A(X) \tag{30}$$

holds, where $C_A(Y)$ denotes the size of the codeword/program for an arbitrary ensemble Y.

Proof. Since $A''\#(\cdot)$ has a *strictly* monotonic dependence as $A\#(\cdot)$ increases, then for every[9] $\epsilon > 0$, there is a sufficiently large $\epsilon' > 0$ such that, if $A\#(X) + \epsilon' \leq A\#(X')$, one has it that $A''\#(X) + \epsilon \leq A''\#(X')$ holds. As a consequence, one can always pick a sufficiently large k_0 and ensembles X and X' where $A\#(X) + k_0 \leq A\#(X')$ such that

$$-\log_2(A''\#(X')) + \mathbf{O}(1) < -\log_2(A''\#(X)) \tag{31}$$

holds. Therefore, Eq (30) follows from the inequality in Eq (24). □

As proposed by [5–7, 9, 10], our results demonstrate that ensembles[10] with higher assembly numbers[11] are more compressible, and therefore they diverge more from those ensembles that are outcomes of (or constituted by) perfectly random processes. A more random generative process for an ensemble (for which the assembly number would score lower) would mean that there are fewer constraints or biases playing a role in this process. In turn, high-assembly-index objects would more rarely appear (or be constructed) because they are more complex (i.e., as we have demonstrated in the previous sections, less compressible or more incompressible)—this is what occurs in scenarios in which there is a weaker presence of *top-down* (or downward) *causation* [26] behind the possibilities or paths that lead to the construction of the objects.

On the other hand, a stronger presence of top-down causation leads to a higher compressibility of the ensemble. This in turn means more constraints and biases playing a role in increasing the probability of high-assembly-index objects. Should one *assume* that the assembly number is indeed capable of measuring this effect of the ensemble as a whole on the assemblage of its objects, then a stronger presence of top-down causation would lead to a higher assembly number.

Thus, under such an *assumption*, we have demonstrated that a more compressible ensemble implies that more constraints or biases played a role in generating more unlikely (i.e., less compressible) objects more frequently than would have been the case in an environment with fewer constraints and biases (i.e., a more random or incompressible environment), hence increasing the frequency of occurrence of high-assembly-index (i.e., less compressible) objects in this environment. Conversely, under the same assumption, if more biotic (or less random abiotic) processes being present in an ensemble imply a higher assembly number, then more biotic processes being present in an ensemble would also imply that the ensemble is more compressible. Therefore, along with Section A.2, the results in this Section A.3.4 demonstrate that in the *general case* (even when the underlying generative process of the ensembles is unknown) *assembly number* $A\#(\cdot)$ is an *approximation* to algorithmic complexity (or algorithmic probability), but a *suboptimal* measure with respect to the more general methods from algorithmic information theory.

In Section A.2.1 we have demonstrated that the assembly number is a defective approximation to algorithmic complexity (see also [8] and Section A.2 for distortions on

[9] Except for the (non-relevant) limit case in which one picks both an ensemble X and a value of ϵ such that $A''\#(X) + \epsilon \geq \sup\{A''\#(X) | X \in \mathbf{X}\}$.

[10] Not to be conflated with the objects themselves.

[11] Not to be conflated with the assembly indexes.

the assembly index). In addition, both the assembly number and the assembly index are defined in such a way that *prevents AT to quantify, even in principle*, the environmental influence and constraints, or the presence of top-down causation. This is because the assembly index (not to be confused with the assembly number mentioned in the above paragraphs) of an object in its current mathematical formulation is not sensible to increases or decreases in the "complexity" of the ensemble as a whole (or, as AT purports, in the skewness toward high-"complexity" objects in the ensemble). The assembly index of an object is a value independent of environmental conditions, and thus an *intrinsic* complexity measure; and any other measure defined upon it (such as the assembly number) can only quantify either individual or collective variations of an intrinsic complexity measure. The assembly number in its current state can only quantify the *global* variations of "complexity" resulting from individual *local* variations of the objects' "complexities", or variations in the distribution of these individual values. However, it cannot quantify the other way around, that is, quantify local variations of "complexity" resulting from global variations of "complexity".

Instead, in order to take into account top-down causation, the assembly index of an object itself would also need to depend on the dynamics or organization of extrinsic factors, e.g. environmental catalytic conditions that may play a role in adding or reinforcing biases toward the formation of higher-assembly-index objects through less likely assembly pathways. Otherwise, the assembly index may classify a low-complexity molecule as being constructed by a more complex extrinsic agent, which is in fact of a much simpler nature (e.g. a naturally occurring phenomenon or abiotic environmental conditions). That is, in case a sufficiently complex environmental catalytic condition plays the role of this extrinsic factor (which increases the bias toward the construction of a more complex molecule), such a level of complexity would be completely missed by the capabilities of a simplistic measure such as the assembly index, thereby rendering it prone to false positives [8, 25].

This kind of influence (whether top-down or bottom-up) in the interplay between global and local scales is e.g. encompassed by complexity measures based on algorithmic probability (or, equivalently, algorithmic complexity) [26–29]: when one approximates the algorithmic probability of either an individual or a collective (e.g., an object or a set), a greater quantity of possible underlying generative processes of different nature (whether intrinsic or extrinsic) are taken into account with the purpose of estimating the most probable candidate generative model. Thus, a pervasive comparison in *future research* between assembly number and algorithmic-probability-based measures for group-controlled ensembles sampled from distinct environments may help quantify the presence of AT's false positives; and help rule out or confirm the presence of top-down causation affecting higher assembly indexes of *abiotic* molecules in higher-assembly-number *abiotic* ensembles. For example, as the assembly number (or, as we have demonstrated in this article, the compressibility) of an abiotic ensemble sufficiently increases for certain environmental conditions, this relative increment of *global* "complexity" may be sufficient to in turn facilitate the formation of abiotic molecules with relatively higher assembly index. By empirically measuring an effect, whether negligible or relevant, on the assemblage of individual objects, *future research* is necessary to quantify the presence of top-down causation—a problem also raised in [30]. Respectively, this study may rule out or confirm its effects on objects, and therefore if the current assembly index formalism itself is corroborated or falsified.

A.4 Lack of empirical evidence in favour of Assembly Theory and presence of empirical evidence against it

The experiments and analysis of organic versus non-organic molecules from mass spectral data using several complexity indexes used as positive evidence for Assembly Theory in [9] empirically demonstrates what we have proven: that the assembly index is equivalent to statistical compression, given that it was not compared to any other data type or any other existing index of the same nature. The simplicity of the experiments required to prove the utility of the assembly index and the main hypothesis of Assembly Theory inclines us to believe that this was a serious omission.

Together with the theoretical results, Fig 1 from [8] shows that the evidence that the AT authors present as being in favour of their theory and algorithms, specifically that their assembly (compression) index may correspond to the way objects assemble, is flawed and invalid. The empirical evidence actually shows that other, equally plausible and valid compression schemes that are sequential, count copies, and build a repeating symbol dictionary tree, are likely the ones from which AT's discriminatory power derives and on which AT's concepts and methods are based.

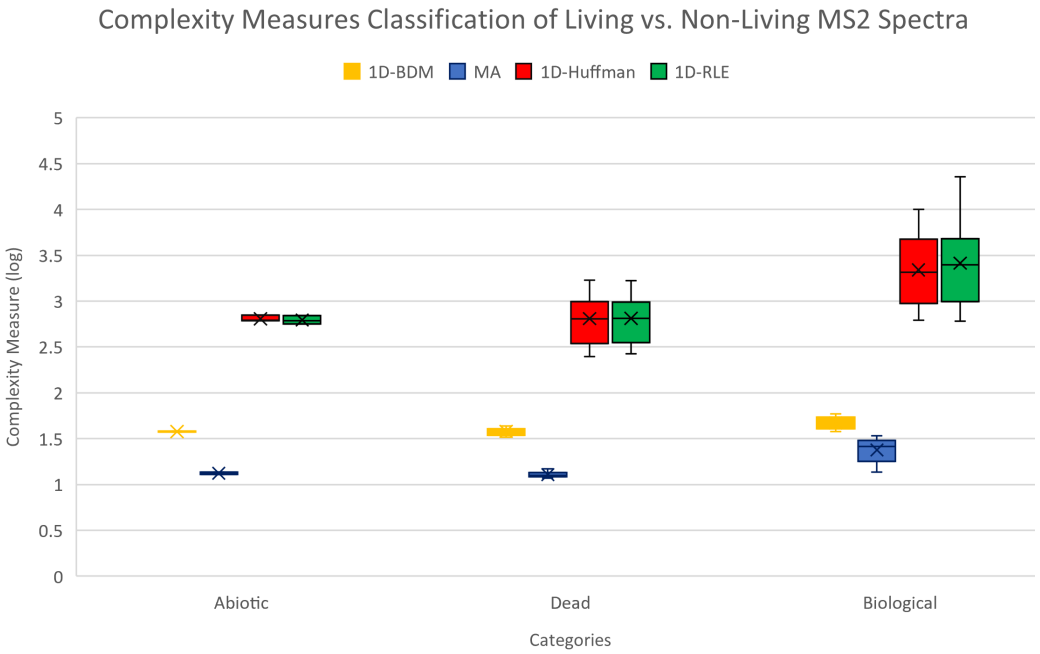

Fig 1. Taken from [8]. The most basic statistical complexity indexes applied to the 18 molecular compounds for which the spectral data was made available from [9]. MA stands for Molecular Assembly based on the assembly index of the molecular compounds. These results were obtained even without optimising the algorithms after a simple flattening and bnarisation procedure of the data (full methods, data and code available at [8]). The results show that other statistical measures separate the data just as MA does (or better) between living and non-living compounds.

The strongest positive correlation in Fig 1, was identified between MA and 1D-RLE coding (R= 0.9), which is one of the most basic coding schemes in computer science, known since the 60s, and among the most similar to the intended definition of MA, as being capable of 'counting copies', together with LZ, which is exactly equivalent to MA or the assembly index. All indexes were applied to spectral data (only 18 extracts available in the original paper in [9]) and taking their results at face value. Other

coding algorithms, including the Huffman coding (R = 0.896), also show a strong positive correlation with MA. The analysis further confirms our previous findings of the similarity in performance between MA and other basic compression measures (that essentially only 'count copies' in data, all of them converging to Shannon Entropy) in classifying living vs. non-living mass spectra signatures. All molecules included in the Assembly Theory paper were included. In other words, both theory and experiment converge, and the experiments confirm with the findings in this paper that Assembly Theory is equivalent to Shannon Entropy and their assembly index is an LZ compression algorithm renamed.

References

1. Chaitin G. Algorithmic Information Theory. 3rd ed. Cambridge University Press; 2004.
2. Calude CS. Information and Randomness: An algorithmic perspective. 2nd ed. Springer-Verlag; 2002.
3. Li M, Vitányi P. An Introduction to Kolmogorov Complexity and Its Applications. 4th ed. Texts in Computer Science. Cham: Springer; 2019.
4. Downey RG, Hirschfeldt DR. Algorithmic Randomness and Complexity. New York, NY: Springer New York; 2010.
5. Marshall SM, Moore D, Murray ARG, Walker SI, Cronin L. Quantifying the pathways to life using assembly spaces. arXiv Preprints. 2019;.
6. Sharma A, Czégel D, Lachmann M, et al. Assembly theory explains and quantifies selection and evolution. Nature. 2023;622(1):321–328. doi:10.1038/s41586-023-06600-9.
7. Marshall SM, Mathis C, Carrick E, Keenan G, Cooper GJT, Graham H, et al. Identifying molecules as biosignatures with assembly theory and mass spectrometry. Nature Communications. 2021;12(1). doi:10.1038/s41467-021-23258-x.
8. Uthamacumaran A, Abrahão FS, Kiani NA, Zenil H. On the Salient Limitations of the Methods of Assembly Theory and Their Classification of Molecular Biosignatures. npj Systems Biology and Applications. 2024;10(1):82. doi:10.1038/s41540-024-00403-y.
9. Marshall SM, Murray ARG, Cronin L. A probabilistic framework for identifying biosignatures using pathway complexity. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2017;375(2109):20160342. doi:10.1098/rsta.2016.0342.
10. Marshall SM, Moore DG, Murray ARG, Walker SI, Cronin L. Formalising the Pathways to Life Using Assembly Spaces. Entropy. 2022;24(7):884. doi:10.3390/e24070884.
11. Kieffer JC, En-Hui Yang. Grammar-based codes: a new class of universal lossless source codes. IEEE Transactions on Information Theory. 2000;46(3):737–754. doi:10.1109/18.841160.
12. Lehman E, Shelat A. Approximation algorithms for grammar-based compression. Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. 2002; p. 205–212.

13. Rytter W. Grammar Compression, LZ-Encodings, and String Algorithms with Implicit Input. In: Díaz J, Karhumäki J, Lepistö A, Sannella D, editors. Automata, Languages and Programming. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004. p. 15–27.

14. Gańczorz M. Entropy bounds for grammar compression. arXiv preprint arXiv:180408547. 2018;.

15. Charikar M, Lehman E, Liu D, Panigrahy R, Prabhakaran M, Sahai A, et al. The Smallest Grammar Problem. IEEE Transactions on Information Theory. 2005;51(7):2554–2576. doi:10.1109/TIT.2005.850116.

16. Cover TM, Thomas JA. Elements of Information Theory. Hoboken, NJ, USA: John Wiley & Sons, Inc.; 2005.

17. Lempel A, Ziv J. On the complexity of finite sequences. IEEE Transactions on information theory. 1976;22(1):75–81. doi:10.1109/TIT.1976.1055501.

18. Ziv J, Lempel A. Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory. 1978;24(5):530–536. doi:10.1109/TIT.1978.1055934.

19. Welch TA. A technique for high-performance data compression. Computer. 1984;17(06):8–19.

20. Billingsley P. On the Coding Theorem for the Noiseless Channel. The Annals of Mathematical Statistics. 1961;32(2):594–601.

21. Salomon D, Motta G. Handbook of Data Compression. London: Springer London; 2010.

22. Kempes C, Walker SI, Lachmann M, Cronin L. Assembly Theory and Its Relationship with Computational Complexity. arXiv Preprints. 2024;(arXiv:2406.12176).

23. Gillen C, Jeancolas C, McMahon S, Vickers P. The Call for a New Definition of Biosignature. Astrobiology. 2023;23(11):1228–1237. doi:10.1089/ast.2023.0010.

24. Malaterre C, Ten Kate IL, Baqué M, Debaille V, Grenfell JL, Javaux EJ, et al. Is There Such a Thing as a Biosignature? Astrobiology. 2023;23(11):1213–1227. doi:10.1089/ast.2023.0042.

25. Hazen RM, Burns PC, Cleaves HJ, Downs RT, Krivovichev SV, Wong ML. Molecular assembly indices of mineral heteropolyanions: some abiotic molecules are as complex as large biomolecules. Journal of The Royal Society Interface. 2024;21(211):20230632. doi:10.1098/rsif.2023.0632.

26. Abrahão FS, Zenil H. Emergence and algorithmic information dynamics of systems and observers. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences. 2022;380(2227). doi:10.1098/rsta.2020.0429.

27. Abrahão FS, Wehmuth K, Ziviani A. Algorithmic networks: Central time to trigger expected emergent open-endedness. Theoretical Computer Science. 2019;785:83–116. doi:10.1016/j.tcs.2019.03.008.

28. Abrahão FS, Wehmuth K, Ziviani A. Emergent Open-Endedness from Contagion of the Fittest. Complex Systems. 2018;27(04). doi:10.25088/ComplexSystems.27.4.369.

29. Abrahão FS, Wehmuth K, D'Ottaviano IML, Carvalho LLd, Zenil H. Expected emergent open-endedness from partial structures extensions under algorithmic perturbations. In: Theoretical and Foundational Problems in Information Studies (TFPIS); 2021.Available from: `https://tfpis.com/`.

30. Jaeger J. Assembly Theory: What It Does and What It Does Not Do. Journal of Molecular Evolution. 2024;doi:10.1007/s00239-024-10163-2.