Why Kolmogorov Complexity in Physical Equations?

Vladik Kreinovich and Luc Longpré
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
emails {longpre,vladik}@cs.utep.edu

Abstract

Several researchers, including M. Gell-Mann, argue that the notion of Kolmogorov complexity, developed in the algorithmic information theory, is useful in physics (i.e., in the description of the physical world). Their arguments are rather convincing, but there seems to be a gap between traditional physical equations and Kolmogorov complexity: namely, it is not clear how the standard equations of physics can lead to algorithmic notions underlying Kolmogorov complexity. In this paper, this "gap" is bridged: we explain how Kolmogorov complexity naturally appear in physical equation.

1 Introduction

1.1 The notion of Kolmogorov complexity

The notion of complexity (usually informal) is very useful in physics. For example, observations that lead to a sequence of all 0's describe a very simple phenomenon, because the resulting sequence can be simply describe as "all zeros". The observations that lead to an oscillating sequence 0101...01 is slightly more difficult to describe, but still reasonably easy. On the other hand, if the observations can only be described by a very complicated system of partial differential equations, then these observations clearly describe a much more difficult phenomenon.

This notion of complexity was formalized in the 1960s by three researchers: G. Chaitin, A. Kolmogorov, and R. Solomonoff. The resulting definition defines a *complexity* $K(x)$ of a sequence of symbols x as the length of the shortest program p that produces the output x. A sequence of symbols is usually called a *word*. By a *program*, we mean a program in a *universal programming* language $\mathcal{U}$ (like C or Pascal or Fortran). Here:

- *programming* language means that we have an algorithm (called *compiler*) that, given a text in the language $\mathcal{U}$, makes some algorithm run on the computer;

- *universal* means that we can, in principle, describe an arbitrary algorithm in this language $\mathcal{U}$.

Thus defined value $K(x)$ is usually called the *Kolmogorov complexity* of the word x.

This definition started an interesting area of research that is often called *Algorithmic Information Theory*. For a modern survey of this research area, see, e.g., [5].

1.2 Kolmogorov complexity is useful in physics

According to its very definition, Kolmogorov complexity describes the algorithmic complexity of different objects. At first glance, this complexity is more relevant for computers than for describing the objective physical world. However, Kolmogorov complexity is very useful in physics as well.

First, Kolmogorov complexity forms the basis for defining the notion of *randomness*, the notion that is central to statistical and quantum physics; see, e.g., [1, 5, 2, 3, 4]. Thus, Kolmogorov complexity is very useful in *foundations* of physics.

Second, several researchers, including M. Gell-Mann, argue, convincingly, that this notion may be useful in *working* physics as well, as an important part of equations that describe the evolution of physical systems [1]. In particular, Gell-Mann shows that Kolmogorov complexity seems to be an appropriate tool for describing biological systems (and complex systems in general).

1.3 The problem with using Kolmogorov complexity in physical equations

The main problem with this idea is that there seems to be a wide logical gap between traditional physical equations and the notions of algorithmic information theory. Because of this gap, adding Kolmogorov complexity to physical equations seems very *ad hoc*.

Let us explain why there is a (perceived) gap. Traditionally, physics considers systems of differential equations that describe how the state of a physical system changes with time. If we know the initial state of the system $s(t_0)$ at some initial moment of time t_0, then we can use these equations to predict the state of the system $s(t)$ at a future moment t. The prediction algorithm normally consists of solving ("integrating") the given system of differential equations. As a result, we get a relation $s(t) = E(t_0, t, s(t_0))$, where E is a computable function of three variables: two real-valued variables t_0 and t, and a (usually multi-component) variable $s(t_0)$ that describes the initial state of the system. This function describes the *evolution* (change in time) of a physical system and is, therefore, sometimes called an *evolution operator*.

As we have already mentioned, Gell-Mann shows that Kolmogorov complexity is an appropriate tool for describing the current state and evolution of different biological systems and other complex systems. On the basic level of evolution equations, Gell-Mann's idea is, in effect, to explicitly add Kolmogorov complexity (e.g., Kolmogorov complexity of the description of the initial state $s(t_0)$), to the equations, so that the next state become algorithmically dependent on t_0, t, $s(t_0)$, *and* also on the Kolmogorov complexity $K(s(t_0))$ of the (description of) the initial state $s(t_0)$.
There is an immediate problem with this idea:

- We have just mentioned that, in traditional physics, the relation E between the original state $s(t_0)$ and the predicted state $s(t)$ is usually *algorithmic*.

- However, it is known that Kolmogorov complexity *cannot* be computed by any algorithm (see, e.g., [5]). Therefore, if we make the dependence E explicitly dependent on the Kolmogorov complexity, this dependence E stops being algorithmic.

This non-algorithmic character of the "evolution operator" E is unusual but *not that problematic*, because many equations of modern physics (especially of modern super-string theories) are so complicated that no general algorithm is known for solving them, and it is quite possible that no such general algorithm exists at all. It is therefore quite reasonable to consider non-algorithmic evolution operators E, i.e., to extend the original class of *algorithmic* evolution operators to some more general class.

A more *serious problem* is that physically natural generalizations of the class of all algorithmic evolution operators do not seem to naturally lead exactly to Kolmogorov complexity, and thus, the emergence of Kolmogorov complexity does not seem to be well related with physics.

1.4 What we are planning to do

The main objective of the present paper is to "bridge" the above gap, and to show that Kolmogorov complexity can indeed naturally appear in physical equations.

2 Which non-algorithmic evolution operators are natural in physics?

2.1 It is quite possible that evolution operators are non-algorithmic

We have already mentioned that it is possible that some physical equations lead to non-algorithmic evolution operators E. A reasonable question is: which non-algorithmic evolution operators can naturally appear in physics?

2.2 General description of physical equations

Most physical equations describe an explicit (and thus, algorithmically checkable) relation between the values of the fields and their derivatives, a relation that must hold for all possible moments of time, and at all possible points in space. The main problem of solving this equation is to find the values of these fields and derivatives that satisfy the given system of equations.

2.3 Simplified case: finitely many space-time events

If each equation contained only *finitely many* conditions, i.e., it must be true in finitely many moment of time and at finitely many points of space, then, for each candidate solution, we would algorithmically check whether this candidate is indeed a solution or not. This possibility would lead, in principle, to an algorithm for solving the given equation.

Crudely speaking, in this algorithm, we enumerate all possible candidate solutions and for each of them, check whether this candidate solution is indeed a solution or not. We stop checking when we find a solution. Of course, this idea needs some (minor) refinement if we want to actually use it:

- In principle, there is a continuum of possible real numbers; therefore, there is a continuum of possible fields etc. This means that we *cannot* simply *enumerate* all possible values of the fields.

- However, we are interested not in the abstract mathematical possibilities, but in the results that can be produced by a computer or at least described on a sheet of paper. Whatever we can store in the computer is a sequence of 0's and 1's, i.e., ultimately, a finite sequence of symbols from a finite alphabet. Whatever we can place on a sheet of paper is, too, a finite sequence of symbols in a finite alphabet. In any given finite alphabet, there are only countably many words of finite length, and therefore, we can, effectively, enumerate (and try) all these words.

2.4 Real-life world: infinitely many space-time events

In real-life physical problems, each equation means the validity of this equality in *infinitely many* moments of time and at *infinitely many* spatial points. These infinities do not necessarily mean that each problem is indeed not algorithmically solvable: it simply means that simply *directly* trying all possible options is no longer possible.

- In most problems of practical physics, we have *indirect* algorithmic solving methods, i.e., methods which do not use the (impossible) exhaustive search. In other words, in these problems, we have an indirect way of checking, in *finitely many* computational steps, whether a given system of equations indeed holds for all *infinitely many* points in space-time.

- On the other hand, starting from the well-known Gödel's theorem, it is known that there are problems in mathematics (and in numerical mathematics) in which no indirect algorithm is possible that would replace the infinite exhaustive search by a finite algorithmic procedure. So far, such problems have not yet been found in physics, but, as we have already mentioned, there is a strong evidence that such problems may occur in physics as well.

How can we describe the resulting possible non-algorithmic evolution operators?

2.5 Towards mathematical formulation of physical non-algorithmic evolution operators

We have already assumed that for every possible candidate solution x, and for every possible point m in space-time, checking whether the candidate m satisfies the given system of equations at this point in space-time, is algorithmically checkable. Let us denote this algorithmically checkable property by $P(x, m)$. Therefore, to check whether a candidate x is a solution to the given system of equations, we must check whether this property $P(x, m)$ is satisfied for *all* possible points in space-time, i.e., whether the formula $\forall m\ P(x, m)$ is true or not.

Thus, if we have an algorithmically non-computable evolution operator, we can "compute" its result if we can detect whether such formulas $\forall m\ P(x, m)$, with algorithmically checkable properties P, are true or not.

To finalize this description, we must describe the set of possible values for the variable m (that describes different points in space-time). In principle, there are *continuum many* possible points m in space-time. However, similarly to the above argument about the world with finitely many space-time points, we can argue that we are only interested in the space-time points that are representable in a computer or at least describable on a sheet of paper. Each such point in space-time can be described by a finite sequence of symbols (even a sequence of 0's and 1's), and therefore, we can, in principle, *enumerate* all of them. Thus, we can assume that the variable m runs over all possible sequences of 0's and 1's.

From the mathematical viewpoint, it does not really matter how we describe these sequences, but from the computer viewpoint, the most useful representation is to interpret each sequence as a non-negative integer, i.e., as a natural number.

Thus, if we have a non-computable evolution operator, we can "compute" it if we can, for every algorithmically checkable predicate $Q(m)$, check whether $\forall m\ Q(m)$ is true or not, where m runs over all possible natural numbers.

Thus, we can describe physically meaningful non-computable functions as follows: they are "computable" by an algorithm that, in addition to normal computer operations, can also ask, for any given algorithmically checkable predicate $Q(m)$, whether the formula $\forall m\ Q(m)$ is true or not. In theory of computation, such "computing" is called *computing with an oracle* (see, e.g., [6]). In these terms, we are interested in functions that are *computable with an oracle that, for a given algorithmically checkable predicate $Q(m)$, checks whether the formula $\forall m\ Q(m)$ is true or not.*

2.6 What we are planning to show

In the following text, we will show that, surprisingly, this class of non-computable functions coincides with the class of functions that are computable with respect to Kolmogorov complexity.

This result bridges the above-mentioned gap by explaining why Kolmogorov complexity naturally appears in physical equations.

3 Main result

Comment. In this section, all the variables run over sequence of 0 and 1, or, equivalently, over natural numbers. Correspondingly, by a *function*, we mean a function from natural numbers to natural numbers, or, equivalently, a function from finite sequences of 0's and 1's to similar finite sequences.

Definition 1. *We say that a function is physically computable if it can be computed with an oracle that, for a given algorithmically checkable predicate $Q(m)$, checks whether the formula $\forall m\ Q(m)$ is true or not.*

Definition 2. *We say that a function is computable relative to Kolmogorov complexity if for some universal programming language $\mathcal{U}$, this function is computable with an oracle that, given a word x, returns the Kolmogorov complexity $K_{\mathcal{U}}(x)$ of this word with respect to this language $\mathcal{U}$.*

Theorem. *A function is physically computable if and only if it is computable relative to Kolmogorov complexity.*

Physical comment. Thus, every evolution operator $E(t_0, t, s(t_0))$, which is physically computable, can be described as a function that algorithmically depends on the inputs themselves (i.e., on t_0, t, $s(t_0)$), and on the Kolmogorov complexity of these inputs.

Mathematical comment. We actually prove a result than is stronger than our Theorem:

- The Theorem states that if a function $f(n)$ is physically computable, then there exists a universal programming language $\mathcal{U}$ such that the function $f(n)$ can be computed by using the corresponding Kolmogorov complexity $K_{\mathcal{U}}(x)$ as an oracle. In this formulation, it is possible that this language depends on the function $f(n)$.

- We actually prove that we can select the universal programming language $\mathcal{U}$ from the very beginning and use this same language for *all* physically computable functions $f(n)$.

4 Proof

4.1 The structure of the proof

The proof consists of two parts:

- First, we prove that every function that is computable relative to Kolmogorov complexity is also physically computable. This is the easier part of the proof.

- Second, we prove that every function that is physically computable is also computable relative to Kolmogorov complexity. This is a more technically complicated part of the proof.

4.2 First part: proof that that every function that is computable relative to Kolmogorov complexity is also physically computable

To prove this result, it is sufficient to show that Kolmogorov complexity itself is physically computable, i.e., that Kolmogorov complexity can be computed with an oracle that, for a given algorithmically checkable predicate $Q(m)$, checks whether the formula $\forall m\ Q(m)$ is true or not.

Indeed, if we have such an oracle, then for every program p, we can check whether this program halts or not (i.e., whether it continues indefinitely without returning any answer at all, or whether it eventually stops and produces some answer). Indeed, for every moment of time t, we can algorithmically check whether this program p has stopped by this time t or not, by simply running the program p for this time t. Thus, the property $S(t)$, meaning that the program stops by time t, is algorithmically checkable. Therefore, the negation $\neg S(t)$ of this property is also algorithmically checkable. Hence, we can use our oracle to check whether $\forall t\ \neg S(t)$ if true or not, i.e., whether the program continues indefinitely or it stops.

Since we are able to check whether each program halts or not, we can compute the Kolmogorov complexity of a given word x as follows:

- First, we try all programs p of length 1. For each of these programs, we check whether this program halts or not.

  - If we conclude that the program p does not halt, we ignore it.
  - If we conclude that the program p does halt, we run it until it halts, and compare its result with x.

  If one of the results coincides with x, this means that $K(x) = 1$, so we can finish our computations. Otherwise, we conclude that $K(x) > 1$.

- Suppose now that we have already checked all programs of length $< \ell$ and none of them generates x. Then, we check all the programs of length ℓ. Again, for each of these programs p, we check whether this program halts or not.

  - If we conclude that the program p does not halt, we ignore it.
  - If we conclude that the program p does halt, we run it until it halts, and compare its result with x.

  If one of the results coincides with x, this means that $K(x) = \ell$, so we can finish our computations. Otherwise, we conclude that $K(x) > \ell$, and try the next possible length ($\ell := \ell + 1$).

Since one of the programs definitely computes x (e.g., the program `write(`x`)`), this algorithm will eventually stop and produce the desired value of the Kolmogorov complexity $K(x)$.

4.3 Second part: proof that every function that is physically computable is also computable relative to Kolmogorov complexity

We will start this second part of the proof by designing a universal language $\mathcal{U}$ for which this result will be proven to be true.

To construct this language, we will start with an arbitrary universal language $\mathcal{U}_0$, and then do the following two-step transformation.

First, we produce an *intermediate* language $\mathcal{U}_1$ in which all programs have even length. Programs from this language have one of the two forms:

- of the type $00p$ and $01p$, where p is a program from the language $\mathcal{U}_0$ whose length is even; and
- of the type $1p$, where p is a program from the language $\mathcal{U}_0$ whose length is odd.

It is clear that this is a universal language because the original language $\mathcal{U}_0$ is universal, and everything that can be computed by a program p from that original language can also be computed by the corresponding program from the language $\mathcal{U}_1$.

The compiler for this intermediate language is easy to write:

- If we get a program, first, we check its length. If it is odd, we return an error message; otherwise, we look at the first character of this program (i.e., 0 or 1).
- If the first character is 1, we delete this 1, and apply the compiler for the language $\mathcal{U}_0$ to the resulting program.
- If the first character is 0, we delete the first *two* characters and apply the compiler for the language $\mathcal{U}_0$ to the resulting program.

Finally, based on this intermediate language, we design the language $\mathcal{U}$ that works as follows:

- If the language $\mathcal{U}$ inputs a program p of *even* length, then it:
  - deletes the first two characters from this program p; and
  - applies $\mathcal{U}_1$ to the resulting shortened program p'.

- If $\mathcal{U}$ inputs a program p of *odd* length, then it:
  - deletes the first character from this program p;
  - applies $\mathcal{U}_1$ to the resulting shortened program p';
  - if the language $\mathcal{U}_1$ halts on p' and produces a word x, it applies $\mathcal{U}_0$ to this word x. If the resulting computations halt too, then x is produced as a result.

Again, it is easy to check that this is a universal language.

Let us show that if we have an oracle that computes the Kolmogorov complexity relative to this universal language, then we can, for every algorithmically checkable predicate $Q(m)$, check whether the statement $\forall m\, Q(m)$ is true or not.

Indeed, since $Q(m)$ is algorithmically checkable, we can design the following algorithm: test the property $Q(m)$ for $m = 0, 1, 2, \ldots$ until you find the value m for which $Q(m)$ is false. Since the original language $\mathcal{U}_0$ is universal, there is a program q in that language that performs the exact same algorithm. This program halts if and only if the statement $\forall m\; Q(m)$ is false. Depending on whether the length of this program q is odd or even, either the program $q' = 00q$, or the program $q' = 1q$ performs the same algorithm in the language $\mathcal{U}_1$.

Let us show that the program q halts if and only if its Kolmogorov complexity $K(q)$ relative to $\mathcal{U}$ is odd. Indeed:

- If $K(q)$ is odd, it means that in the language $\mathcal{U}$, there is a program p of odd length that produces q. According to the definition of $\mathcal{U}$, the only way to have an odd-length program produce anything is when its output is a program that halts. Thus, the program q halts.

- Vice versa, let us show that if the program q halts, then $K(q)$ is odd. Indeed, suppose that we have an even-length program p in the language $\mathcal{U}$ that produces q. By definition of $\mathcal{U}$, this means that if we apply the compiler for $\mathcal{U}_1$ to the program p', that is obtained from p by deleting its first two symbols, we get q. In this case, since q halts, the program $0p'$ also produces the same word q, and the program $0p'$ is 1 symbol shorter than the original program p. Thus, if the word q describes a halting program, then for every even-length program p that generates q, there exists a *shorter* odd-length program that generates the same word. Thus, the shortest program of all programs that generate q is of odd length, i.e., $K(q)$ is odd.

So, if we have an oracle that produces the value of $K(x)$ for every given x, we can:

- form the program q;
- compute the Kolmogorov complexity $K(q)$; and
- check whether this number $K(q)$ is odd or even.

Thus, we will be able to decide whether the program q halts or not, i.e., whether the statement $\forall m\ Q(m)$ is false or true.

The second part of the theorem is thus proven, and so is the theorem itself.

Comment. In the second part of the proof, we have shown that if a function is physically computable, then it is computable with respect to Kolmogorov complexity defined for *some* universal computer. From the *physical* viewpoint, this results answers our question. However, from the *theory of computation* viewpoint, this result raises an interesting open question: Is the above implication true for an *arbitrary* universal language, or only for *some* of these languages?

Acknowledgments

This work was supported in part by NASA under cooperative agreement NCCW-0089, by NSF under grants No. DUE-9750858 and EEC-9322370, and by the Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518.

The authors are thankful to Murray Gell-Mann whose inspiring talk lead to this research.

References

[1] M. Gell-Mann, *The Quark and the Jaguar: Adventures in the Simple and the Complex,* Freeman, N.Y., 1994.

[2] V. Kreinovich and L. Longpré, “Unreasonable effectiveness of symmetry in physics”, *International Journal of Theoretical Physics*, 1996, Vol. 35, No. 7, pp. 1549–1555.

[3] V. Kreinovich and L. Longpré, "Pure Quantum States Are Fundamental, Mixtures (Composite States) Are Mathematical Constructions: An Argument Using Algorithmic Information Theory", *International Journal on Theoretical Physics*, 1997, Vol. 36, No. 1, pp. 167–176.

[4] V. Kreinovich and L. Longpré, "Nonstandard (Non-σ-Additive) Probabilities in Algebraic Quantum Field Theory", *International Journal of Theoretical Physics*, 1997, Vol. 36, No. 7, pp. 1601–1615.

[5] M. Li and P. M. B. Vitányi, *An Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, N.Y., 1997.

[6] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, San Diego, 1994.