

# Kolmogorov complexity, Optimization and Hardness

Yossi Borenstein, and Riccardo Poli

**Abstract**—The Kolmogorov complexity (KC) of a string is defined as the length of the shortest program that can print that string and halts. This measure of complexity is often used in optimization to indicate expected function difficulty. While it is often used, there are known counterexamples. This paper investigates the applicability of KC as an estimator of problem difficulty for optimization in the black box scenario. In particular we address the known counterexamples (e.g., pseudorandom functions, the NIAH) and explore the connection of KC to the NFLTs. We conclude that high KC implies hardness however, while easy fitness functions have low KC the reverse is not necessarily true.

## I. INTRODUCTION

Given a set of feasible solutions ( $X$ ), the object of an optimization algorithm is to find a solution,  $x \in X$  which maximizes (or minimizes) a certain fitness function ( $f : X \rightarrow Y$ ). Complexity theory was developed in order to analyze the relative hardness – i.e., amount of computational resources – of different problems. Usually, computational complexity studies the asymptotic behavior of an algorithm w.r.t. the most difficult or, alternatively, easiest instance of a problem.

An alternative approach to computational complexity focuses on the difficulty of a single instance – irrespectively of the problem for which it belongs. It postulates that the difficulty of a single instance can be measured independently of the problem and thus, the hardness or complexity of a problem – a set of instances – is just an emergent phenomena of the hardness of its instances [15]. This notion of *instance complexity* is one of the many application of *Kolmogorov complexity* (KC), which is also known as, algorithmic information theory, or the principle of *minimum description length* (MDL). Informally, the KC of a binary string is the length of the shortest program that generates it and halts. For example, the string  $\{1\}^n$ , can be represented by the code: "print '1'  $n$  times" and hence, it has a low KC. Since any function (defined over a finite space) can be described by a binary string, the KC of that string is expected to capture the inherent difficulty of the function.

This approach to complexity seems to be particularly interesting to the analysis of general purpose algorithms (also known as metaheuristics [1], black-box algorithms or randomized search heuristics [20]). Metaheuristics are often used when either the problem is not well defined or when there is no sufficient knowledge (or resources) to construct specific algorithms [20]. Since (1) black box algorithms are not designed for any specific problem and (2) the connection between instances and the associated problem for which metaheuristics are applied to is often not clear, it seems appropriate to use the notion of KC (which is defined independently of a problem) when analyzing them.

Indeed, KC was extensively used in this scenario, particularly in the context of the no-free-lunch theorems (NFLTs). In their proof for an almost no-free-lunch, Droste, Jansen and Wegener [4] used the fact that the KC of a repetitive sequence is minimal in order to modify an existing function without changing (almost) its KC – and therefore, its expected difficulty. Schumacher, Vose and Whitley [17] proved that the MDL is not necessarily connected with the NFLTs (i.e., that NFL result can be proven for a set of compressible functions). Streeter [18] proved, on the other hand, that a NFL result *cannot* hold for a particular class of functions which has a *bound* on its MDL. English [6] was able to cast the original NFLT framework into an information framework. He addressed the question of the possible information an algorithm can gain by sampling the search space (see also [22]). The use of algorithmic information allowed English [7] to establish a connection between the complexity of an algorithm and its potential to solve a complex problem. At the same time, observing the high KC of most problems, he argued that even a complex algorithm is not expected to be good in the average case. Finally, Guo et al.[14] suggested using KC as a backbone for a new conceptual framework of GA hardness.

Even though the notion of KC is extensively used, the connection between this measure of complexity and optimization is not clear. In particular, there are known counterexamples which question its usability. The needle-in-a-haystack (NIAH), for example, is a function where the fitness of all the points but the global optimum is 0. The KC of the NIAH is obviously small ( $O(\log(|X|))$ , where  $X$  is the search space), however, it is known to be extremely difficult. Similarly, a trap function, or a function generated by a pseudorandom number generator, have low KC but cannot be optimized efficiently.

In order to understand whether, or whether not, KC is a good conceptual way to measure functions' difficulty we will have first to examine the reasoning that connects it with search. Informally, the logical connection between search, hardness and KC is the following: black box algorithms assume no a priori knowledge about the function being optimized. Rather, they gain knowledge about the function by sampling random points and analyzing the associated fitness values. The more regularities the fitness function contains the easier it is to generalize the data – and hence, the easier it is to locate an optimum point. Kolmogorov complexity or the concept of MDL[11] connects to this notion in a natural way. The more regularities there are the easier it is to *compress* the data, the lower the expected KC is. Since functions associated with low KC are considered easier than those with high KC, *we can equate expected difficulty of a fitness function with*

its compressibility.

The confusion that exists in the literature (on one hand, the use of KC and on the other hand, the known counterexamples) relies in a flaw of the above reasoning: (1) the compressibility as measured by KC does not measure the time-complexity to detect it and therefore, cannot always be associated with regularity, as perceived by search algorithms (2) The existence of regularity does not guarantee efficient search. The objective of this paper is to discuss these issues and to clarify some of the limitation of KC as a measure of problem hardness for optimization.

In section II we will define the KC of a fitness function, explain the scenario of optimization and discuss the general nature of black box algorithms. Section III argues that for any practical point of view, pseudorandom fitness functions are as difficult to search, for any search algorithm as truly random ones. In section IV we define, using KC, a notion of continuity for the fitness function. We argue that KC is not useful for any function which is not continuous in its optima. The relation of the NFLTs and KC is discussed in section V. We show that an inaccurate definition of the KC of a function is the reason for some of the confusion regarding KC. The time-bounded variant of KC is discussed in section VI, we show that these variants are not relevant to our discussion. We conclude with discussion and conclusions.

## II. PRELIMINARIES

In this section we clarify the main issues discussed in the paper. We begin by defining the KC of a string, we then give the most common representation of a fitness function as a binary string – the KC of the function is defined as the KC of that string. We restrict our attention to optimization problems: we assume that: (a) the function has small number of optima points and (b) the only objective is to sample one of these points. We conclude this section by giving a non-usual definition of black-box algorithms, this will make our discussion later in the paper, easier to follow.

### A. Kolmogorov Complexity

The Kolmogorov complexity [13]  $K : \{0,1\}^* \rightarrow N$  is defined as a function from finite binary strings of arbitrary length to the natural numbers  $N$ . It is defined on 'objects' represented by binary strings but can be extended to other types like functions.

The KC of an object,  $x$ , is defined as the length of the shortest program that prints  $x$  and halts. This program can be implemented by any universal programming language (e.g., C++, Java), in which a universal Turing Machine can be implemented. The choice of universal computer (programming language) may change the KC of  $x$  only by a fixed additive constant (which depends on the length of code it takes to simulate one universal computer by another). For this reason, we fix our programming language to an arbitrary language (e.g., Java) and define KC w.r.t. that language. The KC of a string  $x$  is defined as:

$$K(x) := \min_p l(p)$$

where  $p$  is a program that prints  $x$  and halts and  $l(p)$  denotes the length of program  $p$ . This definition suffice for the purpose of this paper – a more accurate definition is given in [13], [12].

A simple or regular object  $x$  has a low KC. For example, a string representing a sequence consisting of  $n$  1s (i.e., "111...1") can be represented by  $K(x) = O(\log n)$  bits. The size of the program depends on the number of bits it takes to encode the number  $n$ . A random object, on the other hand, can only be represented by  $K(x) = n + O(\log n)$ . That is, the shortest program to print  $x$  will be: "print  $x$ ". The KC of an object can only be used as a *conceptual* measure of its difficulty, *it cannot be computed*. That is, it is possible to analyze the properties of a random object, or even to prove that the majority of objects are random but given an object  $x$  it is not possible to prove that it is random.

The notion of KC can be extended to account for functions as well. Let  $f : X \rightarrow Y$  be a mapping between two finite spaces  $X$  and  $Y$ . Let  $I$  be a countable index set. We use  $I$  as a notion of order in  $X$  – that is  $X = \{x_i\}_{i \in I}$ . The Kolmogorov complexity of a fitness function is defined w.r.t.  $I$ . That is, the KC of  $f$  is the complexity of the binary representation of the string  $\{f(x_0), f(x_1), \dots, f(x_n)\}$  (see table I).

TABLE I

THE SETUP FOR CALCULATING THE KOLMOGOROV COMPLEXITY OF A FITNESS FUNCTION. THE SEARCH SPACE IS ORDERED (IN SOME WAY), THE KC IS MEASURED FOR THE STRING DEFINED BY THE  $f(x)$  COLUMN

$x$	$f(x)$	$f(x)$ (Binary)
00000000000000	3	011
00000000000001	4	100
...	...	...
01010101010101	6	110
...	...	...
11111111111111	2	010

Binary Representation of  $f(x)$ : 011100...110...010

Using this notation, the KC of simple versus hard fitness functions will be define w.r.t. that string. For a random fitness function, each co-domain of  $x$  will have to be explicitly defined, and therefore the length of the shortest program  $k(f) = |X| \times \log(|Y|) + O(1)$ . The length of a regular function can be defined as before.

Although this is the common way to define the KC of a function in the EC community (e.g., [6]), this definition is not precise. In section V we will explain the reasons for that and suggest a different definition (based on [13]).

### B. Black-Box Optimization

We would like to focus our attention on functions which cannot be solved efficiently by random search<sup>1</sup> – that is, functions which contain only *small* number of optima points. In order to make this formal, by small, we will mean logarithmic in the size of the search space. Let  $X$  denote the search space, and  $F^{opt} \equiv \{(x_{opt}, f(x_{opt}))\}$  the set of

<sup>1</sup>if this is the case, then the function is, a priori, irrespectively of the search algorithm, easy.

optimum points in the function  $f$ .  $|F^{opt}|$  is bounded above by  $\log |X|$ . In section IV we will use the fact that even if the set of optima points need to be encoded explicitly, its effect on the KC of  $f$  is marginal, i.e., it is bounded above by  $O(\log |X| \times (\log |Y| + \log |X|))$  (which is the number of bits needed to encode explicitly the fitness values of the optimum points).

Randomized search heuristics, as opposed to specialized algorithms, do not use problem-specific knowledge. They sample possible solutions, compute their objective values and accordingly sample additional solutions. This continues until some stopping criteria (e.g., an optima was sampled) is met [5], [20], [21]. Even though it is not exactly the case, it is useful to *think* of a black-box algorithm in the following way:

- 1) Sample random points
- 2) (Estimate accordingly a model for the function  $f$ )
- 3) Estimate accordingly the position of an  $x_{opt}$
- 4) if found  $x_{opt}$  stop, else go back to step 2.

Step 2 can probably be omitted. Most of the existing metaheuristics do not try explicitly to estimate a model of  $f$ . However, Step 3 implies that at least, implicitly they do – how else can they estimate the position of a new point? In any case, this model illustrates how the notion of KC can be connected to problem hardness. Throughout the paper we will use explicitly the notion of *building a model* of the fitness function and examine whether it is helpful or not. While it is possible to think about a model explicitly, it does not change the results of this paper, if instead of a model one will consider the *regularity of the function*.

We assume that the number of queries (i.e., fitness evaluations) made until a solution  $x_{opt} \in F^{opt}$  is found, is used in order to evaluate the performance of the algorithm (see [21], [5] for further discussion).

### III. PSEUDORANDOM STRING

When the KC of a string is greater than or equal to the length of the string, we say that the string is random, it cannot be compressed. A fitness function represented by a random string contains no regularities, no information and therefore, cannot be solved efficiently by any search algorithm.

KC, however, is not the only way to define a notion of randomness. An alternative approach to randomness suggests that a string is *pseudorandom* as long as no *efficient* observer can distinguish it from a uniformly chosen string of the same length. Randomness according to this notion is relative, it is not necessarily an inherent property of the object. It is helpful to consider the following mental experiment given by Goldreich:

“Alice and Bob play head or tail in one of the following four ways. In all of them Alice flips a coin high in the air, and Bob is asked to guess its outcome before the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess. In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with

probability  $1=2$ . In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is determined in principle by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability  $1=2$ . The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate information on the coin’s motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess. In the fourth alternative, Bob’s recording equipment is directly connected to a powerful computer programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.” [9]

Having an infinitely powerful computer, using this analogy, KC is the only way to define randomness. However, when the computer is required to be efficient, any function which cannot be modeled in *feasible* time is as random, effectively, as a truly random function. Since this definition is not absolute (it depends on how *feasible* is defined) we call such strings pseudorandom.

The exact definition of *feasible computation time* is still open to a debate. However, computations that are polynomial-time in the size of the input are usually considered feasible – whereas others (i.e., non-polynomials) are considered to be intractable. Restricting ourselves to efficient algorithms, we can now define formally when two objects (i.e., in our case fitness functions) are indistinguishable.

*Definition 1:* Let  $X_n, Y_n$  be random variables distributed over bit-strings of length polynomial in  $n$ .  $X_n, Y_n$  are computationally indistinguishable if for every probabilistic polynomialtime algorithm,  $A$ , every positive polynomial  $p(\cdot)$ , and all sufficiently large  $n$

$$|Pr_{x \sim X_n}[A(x) = 1] - Pr_{y \sim Y_n}[A(y) = 1]| < \frac{1}{p(n)}$$

where we denote by  $Pr_{x \sim X_n}[A(x) = 1]$  the probability that  $A(x) = 1$  when  $x$  is distributed according to  $X_n$ .

This definition simply states formally that if no program in polynomial time (of the size of the input) can distinguish between a sample of two distributions, effectively these distributions are the same. In other words, we can say that, for example, the problems MAXSAT and Job-Shop-Scheduling (under the same representation, and given the same size) are the same if, given a typical instance, no algorithm can decide in polynomial time to which of the two problems it belongs.

A pseudorandom sequence is defined w.r.t. a uniform distribution. In other words, if  $U_n$  is the uniform distribution taken for bit-strings of size polynomial in  $n$ ,  $X_n$  (defined similarly) is pseudorandom if  $U_n$  and  $X_n$  are indistinguishable in polynomial time. In our case (note that the expected

KC of a string generated by any of the two distributions might be different). From any practical point of view, a fitness function corresponding to a pseudorandom sequence is as hard to optimize as a random fitness function.

#### IV. KOLMOGOROV CONTINUITY

The last section pointed out that, sometimes, even if a function is highly regular (and hence have a low KC) – this regularity cannot be inferred sufficiently fast, and hence for any practical purpose it is of no use. In this section we argue that even if it is possible to construct a model of the function in an efficient way, this does not guarantee, for optimization problems, an efficient search.

Our argument relies on the fact that the number of optima points, as we argued in section II, is expected to be very small. Nevertheless, finding these points is the only objective that an optimization algorithm has. KC measures the regularity of the *entire* function. The effect of a small finite set of points on the complexity as measured by Kolmogorov is very small – and this is the problem. First, even when a formulation of a function is given, it is not necessarily easy to identify points with high fitness values. Second, and this is the reason for which we decided to choose the term *Kolmogorov continuity*, if the set  $\{x_{opt}\}$  which corresponds to all the optima points in the search space is finite (and small), it can be coded explicitly with a table (e.g., *set*  $x_k = opt$ ) and hence, on one hand, have almost no effect on the KC of the function and on the other, is not bounded to be structurally related to the rest of the landscape. We will consider two cases, in the first case, the search algorithm is led to believe that the function is flat – and therefore, there are no optima points. In the second, the algorithm is led to believe that the optimum is at one place, while it is in another. All of our examples are taken from table II.

TABLE II

FUNCTIONS WITH LOW KC WHICH ARE DIFFICULT TO OPTIMIZE (WITH THE EXCEPTION OF 1 AND 2). WE ASSUME THAT  $x \geq 0 \in N$  AND  $n = |X|$  IS THE SIZE OF THE SEARCH SPACE.

$$f(x) = 0 \quad (1)$$

$$f(x) = x \quad (2)$$

$$f(x) = \begin{cases} x - 10 & \text{if } x > 100, \\ f(f(x + 11)) & \text{if } x \leq 100 \end{cases} \quad (3)$$

$$f(x) = \begin{cases} n & \text{if } x = x_{opt}, \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

$$f(x) = \begin{cases} n + 1 & \text{if } x = x_{opt}, \\ x & \text{otherwise} \end{cases} \quad (5)$$

Irrespective of  $|X|$ , the size of the search space, all the functions in table II have a very low KC. Nevertheless, equations 1 and 2 which represents flat and unimodal landscapes respectively, are known to be easy, whereas equation 4, the NIAH is hard and equation 5, a deceptive function is even harder. The difficulty of equation 3 is not clear.

Having formulating a model, even a precise one, of the fitness function is not enough. The model needs to be simple enough to predict, easily, the position of an optimum point. All the equations in table II apart from equation 3 are simple in this sense (i.e., the optimum is either given explicitly or can easily be inferred). Finding the optima, however, of the recursive function (Eq. 3), especially in  $\{1..100\}$ , is far from being trivial. In fact, calculating the function value of *any* point is far from being efficient. In order for the regularity of the function to be useful for optimization it has to be formulated in a useful way. Incidentally, equation 3 is known as the McCarthy 91 function, it can be written in the following way:

$$f(x) = \begin{cases} 91 & \text{if } 0 < x \leq 101, \\ x - 10 & \text{otherwise} \end{cases}$$

Given the alternative representation the position of the optimum is obvious. Unfortunately, more often than not, the simple formulation (if existed) is not known.

The previous example considered a precise model of a fitness function which does not indicate, in a simple way, the position of optima points. In practice, most chances are that the estimated model of the function is noisy. This by itself is not necessarily a problem. Usually, a good model accounts for the *majority* of the observed data (especially when considering some noise). The situation changes, however, when some of the points are very important (e.g., in our case, optima points) whereas other (the majority) do not contribute (at least, not directly) to performance.

The NIAH (equation 4) is a very good example. First, the KC of this function is obviously low: it takes  $O(\log |X|)$  bits to return  $n$  for the global optimum and 0 for all the other solutions in the search space. Yet, it was proven by Droste et al.[5], that the expected performance of any black-box algorithm on the NIAH is  $(|X| + 1)/2$  – that is the expected performance of a random search.

This happens because the global optimum of the NIAH is not *continues* w.r.t. the rest of the landscape. Any model based on statistical estimation of the fitness function *cannot* capture discontinuity. If the discontinuity occurs in the position of the global optimum, the fitness function, irrespectively of its KC, smoothness or any statistical estimation<sup>2</sup> will be difficult to optimize.

The complexity of a fitness function which is not continuous in its optima points cannot be accounted for by any model which is derived from statistical inference. The MDL or KC is meaningless, in this case<sup>3</sup>.

<sup>2</sup>We refer to any statistical estimation which do not use an a priori knowledge regarding the position of the global optimum.

<sup>3</sup>Interestingly, in [2] we explicitly associated flat landscapes with the random decisions an algorithm makes. We suggested a different way to measure the KC of a fitness function based on the way an algorithm interpret it. We focused on algorithms that use tournament selection – in the simplest case, given two alternative solutions the algorithm selects the one with the highest fitness. If the two solutions have the same fitness value, the algorithm selects one randomly. Based on this selection mechanism, any two solutions with the same fitness, corresponds, potentially, to a random decision the algorithm makes. Considering all such decisions, a flat landscape corresponds to a string with maximal KC.



we choose to encode the fitness function into a binary string. Formally, assuming that  $f : X \rightarrow Y$  where  $X$  and  $Y$  are both finite, the KC of  $f$  is defined:

$$K(f) = \min_{p_f \in \{0,1\}^*} \{l(p_f) : \forall x \in X \ p_f(x) = f(x)\}$$

where  $p_f$  is a program that can receive an input  $x$  and return the value  $f(x)$ . That is, the program emulates the behavior of the function, it does not output a string corresponding to the fitness values of all the solutions in the search space. Now, let us assume that the search space  $X$  can be ordered in a natural way  $X = \{x_i\}$  (i.e., any compressible way, e.g., lexicographic) and that the permutation  $\pi$  denotes the particular order in which we want to output the fitness values then, given that program  $p_f$ , we can define the following short procedure that prints the values in the requested order:

for  $i = 1$  to  $|X|$  print( $p_f(\pi(x_i))$ )

Going from the first definition to the second seems to be trivial, however, it is in the definition of  $\pi$  where the complexity of the original problem can be hidden. Remember that, on principal, it is possible to transform, choosing the right representation,  $\pi$ , a random representation of the function, (e.g.,  $S_l$ ) to a regular one (e.g.,  $S_s$ ). However, in order to do so,  $\pi$  will have to be a random function (otherwise, by using  $\pi^{-1}$  we can emulate  $S_l$  using a short code which is in contradiction to  $S_l$  being random). The short procedure, irrespectively of the true  $K(f)$  will have the maximum KC. If, however,  $\pi$  is compressible, then the procedure is constant and hence its effect on the KC of the function is constant as well.

#### A. Representation and Approximation of KC

Interestingly, the previous result suggests that the choice of any (real-world) representation is irrelevant to the expected performance. In particular, any real-world representation is short, and hence, can effect the KC of a function by no more than an additive constant.

However, it is well known that the choice of representation plays a major role in the design of efficient metaheuristics. In particular, metaheuristics work well on neighborhood structures in which close solutions have close fitness values [16]. Since for each problem one can choose different neighborhood structures, this is the best (a priori, known) criteria to select the neighborhood structure for the problem at hand.

So, while KC is invariant to *compressible* representation the performance of metaheuristics is definitely not. It is important to understand that KC is a very indirect way to encode the regularities of a fitness function. In particular, our intuition as to what a regular string is (e.g., the string "0101...01" is regular because it contains a continues repetition of "01") fails when trying to capture the regularity of more complicates examples – the binary representation of a function as defined in section II being one them.

In particular, we argued that the complexity of a function should be effected by the choice of the neighborhood structure – or distance function. The regularity of a problem in the

black box scenario depends on the correlation between the fitness function and the neighborhood structure. How can a program capture the regularity of a function without knowing the distance function? KC is not about *how* a string can be compresses, it is about the *theoretical existence* of a short representation. So, there exist a program that use the the distance function  $d$  in order to compress the data. Since most of the distance functions are compressible (at least the ones used in practice) this does not affect the KC of the program.

KC is a good conceptual way (for optimization) to study the general behavior of search algorithms. However, the problem arises when one tries to approximate the KC of a particular function using, for example the LEMPEL-ZIV compression algorithm. Generic compressing algorithms will not be able to capture the regularities of a problem unless the binary representation of the function is carefully chosen. As we've seen in section II, the complexity of a function is measured w.r.t. a particular *arbitrary order* of the search space – and hence, it cannot capture the regularities of the function. Unless the order of the search space is carefully chosen, we do not believe that it will have any meaningful connection to the complexity of the function.

So, how *should* we order the search space so the complexity of the function will be properly represented? Well, the regularity of the function, at least in the black-box scenario, is usually, by construction, captured only given the distance function, or the metric space. There is no simple order of the search space which can capture the regularity of all representation.

To conclude, as long as the representation is not chosen – all the functions have the same a priori Kolmogorov complexity – simply because different choices with regard to the order of the search space can change a random function to a highly smooth one. However, once a representation is chosen, it is possible to distinguish between compressible functions to incompressible ones. In any case, only a random (i.e., incompressible) permutation of the search space can transform a random function to a regular one.

## VI. TIME BOUNDED KC

There are many variants to the basic notion of KC (the one used in this paper). The time-bounded variants are of particular importance to computational complexity [8]. They explicitly define a short program with a particular bound on its runtime. Instance complexity, for example, was introduced by Orponen et al.[15] – it defines the complexity of a string  $x$  w.r.t. a set  $A$  and time bound  $t$  as the size of the smallest program that runs in time  $t(n)$ , decides correctly whether  $x$  is in  $A$ , and makes no mistakes on other strings (it allows to output the symbol  $\perp$  for "don't know").

$$ic^t(x : A) = \min_p \begin{cases} (1) \text{ For all } y, U(p, y) \text{ runs in time } t(|y|), \\ (2) \text{ For all } y, \text{ if } U(p, y) = 1 \text{ then } y \in A, \\ (3) \text{ For all } y, \text{ if } U(p, y) = 0 \text{ then } y \notin A, \\ (4) U(p, x) \neq \perp \end{cases}$$

The notion of instance complexity is very powerful, in particular it is possible to characterize the different complexity classes w.r.t. the complexity of their instances. Moreover, the notion of instance complexity, or more generally, time-bounded KC, seems to resolve all the problems introduced in this paper (i.e., there is no conceptual problem, to the best of our knowledge, with the *time-bounded* KC as a measure of function difficulty).

However, these variants do not relate to the direct properties of a function, instead, they simply *state* that a problem (function in our case) is easy if it is easy (i.e., there exist an efficient small algorithm that solves it). This is done without suggesting any particular property of the function that makes it easy. While a function with high KC is expected, always, to be difficult to optimize, a compressible function is not necessarily easy. The problem with the time-bounded KC is that it does not explain why.

This is, perhaps, the reason for which most of the work connected with KC and optimization consider only the original definition. In particular, only this definition enables, for example, to construct easy functions or to use LEMPEL-ZIV compression to measure expected difficulty. To conclude, time bounded KC does not explore the relation between compressibility and hardness, and hence is not relevant to the investigation presented in this paper.

## VII. DISCUSSION

Kolmogorov complexity is an important concept with many important applications. For a brief review on the particular importance of KC to Computational Complexity see [8]. In the EC community it is used mainly as a way to approximate (theoretically) the difficulty of a function. In this paper we investigated the applicability of KC for that case.

We started with the logical connection between compressibility – defined by KC – and the expected difficulty to optimize a fitness function. The connection is straightforward, it is assumed that regular functions are easier to optimize, and since compressibility (although in a different way) measures how regular a function is, it can be associated, in a sense, with its expected difficulty.

This proposition, naturally, has two consequences – functions with high KC, contain no structure, no regularity and therefore are expected to be difficult to optimize. Functions, on the other hand with low KC are expected to be easy. The first proposition is indisputable, however, while easy functions have low KC, difficult functions do not necessarily have high KC.

In section III we argued that a fitness function generated by a pseudorandom number generator has a low KC, however, it is expected to be hard to optimize. It can be argued that pseudorandom numbers are not proven to exist. Indeed, pseudo-random numbers generators generate random “*looking like*” long sequence of bits from a short sequence of a real random sequence. The existence of pseudorandom strings is based on the conjecture that  $\mathcal{P} \neq \mathcal{NP}$ . This in turn suggests that there are computational problems which are

inherently intractable. In particular, the existence of random-number generators depends on the existence of One-Way functions – functions which are easy to calculate but hard to invert.

*Definition 2:* A function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is called one-way if the following two conditions hold:

- 1) easy to compute: the function  $f$  is computable in polynomial-time.
- 2) hard to invert: for every probabilistic polynomial-time machine,  $M$ , every positive polynomial  $p(\cdot)$ , and all sufficiently large  $n$ 's

$$Pr_x[M(1^n, f(x)) \in f^{-1}(f(x))] < \frac{1}{p(n)}$$

where  $x$  is uniformly distributed in  $\{0, 1\}^n$ .

Unfortunately, it is not known whether one-way functions exist – in fact this will prove that  $\mathcal{P} \neq \mathcal{NP}$ <sup>5</sup>. However, almost the entire field of cryptography is based on this conjecture [10] and so we do expect it to suffice to our purpose.

Section IV shows that even the existence of a short (detectable) description might not be enough. First, we argued that the description must allow to efficiently estimate the position of an optimum solution – we gave the recursive function as a counterexample. The second and main part of this section is about the limitation of any statistical method to estimate the regularity of a function. We defined a notion of continuity in the function and argued that as long as the optima points are not continuous, it is not possible to estimate the efficiency of the search. We gave two examples.

The NIAH is a known example to a hard function with a short description length. The discontinuity of the *needle* w.r.t. the rest of the landscape explains this contradiction: the KC (or MDL) is not applicable for this case. This clarifies the results obtained by Schumacher et al.[17] which illustrated that the NFL results are “independent from whether or not the set of functions (over which a No Free Lunch result holds) is compressible”. Even though this is true, the kind of compressible set of functions for which a NFL result holds is expected to be difficult to optimize.

The same reasoning (i.e., discontinuity in the optimum) applies for trap or deceptive functions as well. However, they exemplify additional important principle. KC measures how *random* a function is – that is, the worst possible performance, according to this measure is that of random search. The NFLTs [21] prove, however, that for any problem on which an algorithm performs well (i.e., better than random search) there is a problem on which it performs badly (worse than random search). Deceptive functions are well known examples for realistic search algorithms. Indeed, we can equate regularity with bias, and if the optimum is discontinuous, that is, do not follow the regularity of the function, the algorithm is expected to perform worse than random search. This can happen only for very *regular* functions, that is, functions

<sup>5</sup>Note that it is not known whether  $\mathcal{P} \neq \mathcal{NP}$  implies the existence of one-way functions

with *low KC*. Thus we argue that low KC is a condition for performance which is *different*, whether better or worse, than random search.

In section V we argued that by choosing a different representation, we can transform a highly complex, random function to a very easy, regular one. We then showed that this is a consequence of an inaccurate definition. We suggested a new definition for the KC of a function and linking it to the previous definition, we argued that the complexity of the representation has to be taken into account as well. We concluded that section by pointing out that any approximation of the KC by general compressing algorithms is not likely to be of any use when the fitness function is encoded as suggested in II.

Finally, in section VI we considered the time-bounded variants of KC. While these variants do not suffer from the same problems of KC, they do not address particular properties of a fitness function that makes it either hard or easy to optimize. For this reason they are not relevant to our discussion.

### VIII. CONCLUSION

In this paper we investigated the applicability of KC as an estimator of problem difficulty for optimization in the black box scenario. High KC implies hardness. However, while easy fitness functions have low KC the reverse is not necessarily true: (1) Pseudorandomness exemplifies one of the fundamental limitations of KC in relation to optimization: it is not related to time. (2) We defined a notion of Kolmogorov-continuity, we showed that as long as the optima points of a function are not continuous, KC is not expected to be accurate. We gave the NIAH as an example. (3) All the functions on which an algorithm is expected to perform *worse than random search* have low KC. KC measures *the level of randomness* of the function, irrespectively of the expected difficulty. Other than that, the relation of KC to the NFLTs was explored, a new definition of KC of fitness functions was given (based on [13]), and finally, we suggested to avoid any approximation (by general compressing algorithms) of the KC of fitness functions.

### REFERENCES

- [1] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.
- [2] Y. Borenstein and R. Poli. Information landscapes. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1515–1522, New York, NY, USA, 2005. ACM Press.
- [3] E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowland, N. Jonoska, and J. F. Miller, editors. volume 2724 of *Lecture Notes in Computer Science*. Springer, 2003.
- [4] S. Droste, T. Jansen, and I. Wegener. Optimization with randomized search heuristics - the (a)nl theorem, realistic scenarios, and difficult functions. *Theor. Comput. Sci.*, 287(1):131–144, 2002.
- [5] S. Droste, T. Jansen, and I. Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Electronic Colloquium on Computational Complexity (ECCC)*, (048), 2003.

- [6] T. M. English. Optimization is easy and learning is hard in the typical function. In A. Zalzalá, C. Fonseca, J.-H. Kim, and A. Smith, editors, *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*, pages 924–931. IEEE Press, 2000.
- [7] T. M. English. Practical implications of new results in conservation of optimizer performance. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. M. Guervós, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VI, 6th International Conference, Paris, France, September 18-20, 2000, Proceedings*, volume 1917 of *Lecture Notes in Computer Science*, pages 69–78. Springer, 2000.
- [8] L. Fortnow. Kolmogorov complexity and computational complexity, 2004.
- [9] O. Goldreich. Pseudorandomness. In *Automata, Languages and Programming*, pages 687–704, 2000.
- [10] O. Goldreich. Foundations of cryptography a primer. *Foundations and Trends in Theoretical Computer Science*, 1(1), 2005.
- [11] P. Grünwald. A tutorial introduction to the minimum description length principle. *CoRR*, math.ST/0406077, 2004.
- [12] P. Grunwald and P. Vitanyi. Shannon information and kolmogorov complexity. *IEEE Transactions on Information Theory*, 2004. In Review.
- [13] P. Grunwald and P. Vitanyi. Algorithmic information theory. In *Handbook on the Philosophy of Information*. Elsevier, to appear.
- [14] H. Guo and W. H. Hsu. GA-hardness revisited. In Cantú-Paz et al. [3], pages 1584–1585.
- [15] P. Orponen, K. i Ko, U. Sch&#246;ning, and O. Watanabe. Instance complexity. *J. ACM*, 41(1):96–121, 1994.
- [16] P. M. Pardalos and M. G. C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, 2002.
- [17] C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In L. Spector, E. D. Goodman, A. Wu, W. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 565–570, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.
- [18] M. J. Streeter. Two broad classes of functions for which a no free lunch result does not hold. In Cantú-Paz et al. [3], pages 1418–1430.
- [19] N. K. Vereshchagin and P. M. B. Vitányi. Kolmogorov's structure functions and model selection. *IEEE Transactions on Information Theory*, 50(12):3265–3290, 2004.
- [20] I. Wegener. Towards a theory of randomized search heuristics. In B. Rován and P. Vojtás, editors, *MFCS*, volume 2747 of *Lecture Notes in Computer Science*, pages 125–141. Springer, 2003.
- [21] D. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1):67–82, 1997.
- [22] D. P. Y.C. Ho. Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications*, 115(3):549 – 570, 2002.