Kolmogorov Complexity and Distance Sets: Two Notions of Set Complexity

Elana Elman

April 26, 2024

Abstract

I introduce Kolmogorov complexity and the Erd•s distinct distance problem, describe an intuitive connection between these topics, and explore whether they are actually related. I construct sets in $\mathbb{R}^2$ with arbitrary Kolmogorov complexity and distance set size in order to show that the Kolmogorov complexity and the size of the distance set for nite sets of xed size are independent quantities. I consider what alternative descriptions of set complexity might agree more closely with my geometric intuition.

1 Introduction

I am interested in capturing the \structuredness" of a nite set of points. I study two notions of complexity, Kolmogorov complexity and the size of the distance set. Kolmogorov complexity is the length of the shortest program that outputs a given nite string; in the context of sets, I consider

Definition 2.3 (Big-O)**.** Given two functions f and g: $\mathbb{R} \mapsto \mathbb{R}$, f is said to be $\in O(g)$ or $= O(g)$ if $\exists C \in \mathbb{R}$ and $x_0 \in \mathbb{R}$ such that

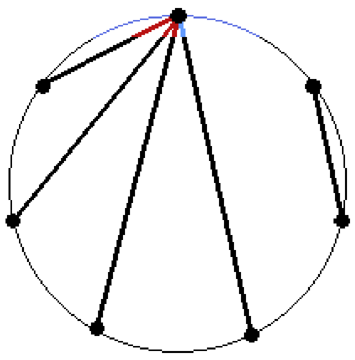
$$|f(x)| \leq Cg(x) \forall x > x_0:$$

Erdős' original paper [1] shows that $M(n)$ is between $O(\sqrt{n})$ and $O(\frac{n}{\sqrt{\log n}})$. The lower bound was gradually improved until 2015, when Guth and Katz proved $M(n)$ is at least $O(\frac{n}{\log n})$.

2.0.1 Upper Bounds

Example. Let P be n random points in the unit square. It is overwhelmingly likely that all distances are unique and $|\Delta(P)| = \binom{n}{2} \in O(n^2)$.

Example. Let P be n points arranged evenly on a circle. Pick any of these points to call a. a is the same distance from each of its neighbors, and the same distance from the second point on its right as from the second point on its left, etc: all distances from a to other points in P come in pairs, except for the single distance from a to the point directly opposite it if n is even. By symmetry, any distance between non-a points is the same as some distance involving a. So $|\Delta(P)| = \lfloor \frac{n}{2} \rfloor \in O(n)$.



- Even the green triangle contains some repeated distances. Any triangle where all three side lengths are integers $\leq \sqrt{n}$ is present within the green triangle, and any integer distance $\leq \sqrt{n}$ also occurs along the top edge of the grid, so the hypotenuse of such a triangle repeats a distance found along the top. The purple lines marked on Figure 2 are two different lines of length 5: one sits on the top of the grid and the other is the hypotenuse of a 3-4-5 triangle.

  Sets of integers $x; y; h$ with $x^2 + y^2 = h^2$ are called Pythagorean triples. Any Pythagorean triple is of the form

$$\begin{aligned} x &= k(2ab) \\ y &= k(a^2 - b^2) \\ h &= k(a^2 + b^2) \end{aligned}$$

  where k is any positive integer and $a > b$ [2]. Restricting to $k = 1$, h is a repeated distance in the grid when $2ab \leq \sqrt{n}$ and $a^2 - b^2 \leq \sqrt{n}$. The number of integer grid points in this region is $O(\sqrt{n}\log n)$[1], so the number of distinct distances in the grid is at most $O(n - \sqrt{n}\log n)$.

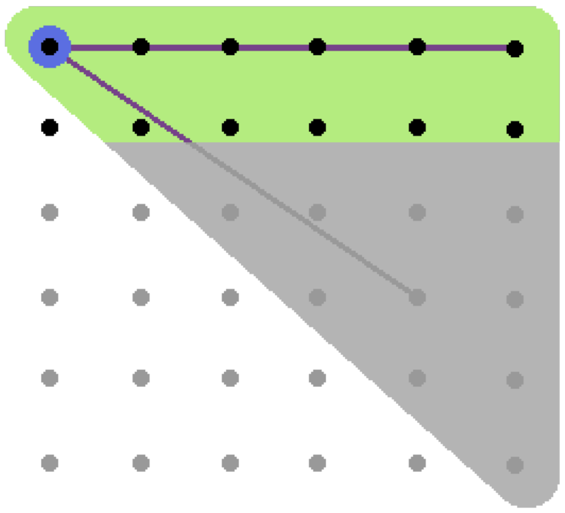


Figure 2: A 6×6 square grid. All distances outside the green triangle are also present in the green triangle. The green triangle contains some repeating distances: two lines of length 5 are drawn in purple.

2.0.2 Lower Bounds

Finding any lower bound on M is less obvious. Here is Erdős' $\sqrt{n}$ bound:

Proof. Let P be any set of n points in the plane. Take any $a \in P$. For each other point p_i in P, draw a circle through p_i centered at a. Call the number of distinct circles drawn m.

- If $m < \sqrt{n}$, some circle must have at least

$$\frac{n}{m} \geq \frac{n}{\sqrt{n}}$$

[1] $n=p$

points, so some hemisphere has at least^{p}

Time	Tape	Current State	Next State	Write	Move
0	:::B $\underline{1}$01B:::	SEEK	SEEK	1	R
1	:::B 1$\underline{0}$1B:::	SEEK	SEEK	0	R
2	:::B 10$\underline{1}$B:::	SEEK	SEEK	1	R
3	:::B 101$\underline{B}$:::	SEEK	CARRY	B	L
4	:::B 10$\underline{1}$B:::	CARRY	CARRY	0	L
5	:::B 1$\underline{0}$0B:::	CARRY	DONE	1	L
6	:::B $\underline{1}$10B:::	DONE			

Table 1: An example run of the increment machine on the input 101

Turing machines for any particular problem aren't unique. In fact, it's possible to translate any Turing machine into a machine with only two states by increasing the number of symbols, or into a machine with two symbols by increasing the number of states. However, the product of the number of states in symbols stays within a constant factor after either of these translations, so we can comfortably use this state $\times$ symbol complexity to describe any Turing machine's complexity.

3.1.2 Programming Languages

Compared to Turing machines, real computers have additional features such as random-access memory and limitations such as nite memory. Turing machines are also di cult to design and understand. Programming languages allow simpler, human-readable descriptions of algorithms. Almost all programming language are Turing-complete, meaning they are capable of simulating Turing machines. A language is called Turing-equivalent if a Turing machine is capable of simulating running its programs, and nobody has ever found a Turing-complete language which is not Turing-equivalent. It seems extremely likely that no programming language can be more powerful than Turing machines.

For a Turing-equivalent language L, let T_L be a Turing machine that simulates a program written in L, and let L_T be a program in L that simulates execution of a Turing machine. Then language L_1 can simulate a program written in language L_2 by simulating the execution of T_{L_2}. Since these programs and Turing machines are nite, there exist nite translations between Turing-equivalent languages. This shows that nite compilers { programs that translateTuring machines are nite, there existely that no an

[illegible]

De nition 3.5 (Universal Functions). A function f is universal for a set of functions F if f minorizes all functions in F.

If f and g are both universal for F, then their di erence on any x is bounded by a constant, and any non-universal function h in F is no more than a constant less than f. This means we can consider only optimal descriptions relative to universal functions without worrying that other functions provide shorter descriptions.

De nition 3.6 (Computable Functions). A partial function $f : \mathbb{N} ! \mathbb{N}$ is called computable if there is some Turing machine which terminates with output $f(n)$ on each input n for which f is de ned.

Theorem 1 (A universal computable function f_U exists). *Proof.* Let U be a Turing machine which simulates other Turing machines. U must take two pieces of information, a description of the machine to simulate and the input string to simulate running on. U expects this input in the format $11...10\ gp$: this is $l(g)$ 1s, then one 0, then the literal description g of the desired Turing machine, then the literal input string p. U may then use the pre x of 1s to separate g from p and run its simulation. Let f_U be the function executed by U. □

De nition 3.7 (Kolmogorov Complexity). The Kolmogorov complexity of a nite string s is de ned as $C(s) = C_{f_U}(s)$. We require that the program doesn't take arguments, or equivalently that any input to the program counts toward its length.

Theorem 2. Most strings are incompressible.

Proof. The set of n-length binary strings has 2^n elements, and the set of shorter strings has 2^{n-1} elements, so there are at most 2^{n-1} di erent outputs of shorter programs. Even if we optimistically assume that each of these outputs has length n, at least $2^n - 2^{n-1} = 2^{n-1}$ of the 2^n strings with length n cannot be more concisely represented. □

Theorem 3. Kolmogorov complexity is uncomputable.

Proof. Suppose a program K with length a takes as input a nite string S and returns its Kolmogorov complexity. Note that some shortest program always exists because it is at longest $\log S$.

Write a new function C which decides if a string S is compressible:

```
from math import log
def C(S):
    if K(s) >= log(s, 2):
        return false
    return true
```

This program is only a constant number of bits c_B longer than C. B is $a + c_C + c_B$ bits long.

Choose $min = 2^{a+c_C+c_B}$. Now $B(min)$ returns a number b with $K(b) > 2^{a+c_C+c_B}$. On the other hand, we just saw that b was generated by the program $B(min)$, which, including the length of its argument, was at most $a + c_C + c_B + \log 2^{a+c_C+c_B} = 2(a + c_C + c_B)$. This is a description of b which is shorter than than $K(b)$, the shortest possible description. This is a contradiction, so the function K cannot actually exist.

□

The above definitions and facts are from Li and Vitanyi's book on Kolmogorov complexity [3], except for the proof of uncomputability, for which I referenced Peter Miltersen's course notes [5]. The below definition is my own, for purposes of comparison with the Erdős distance problem:

Definition 3.8 (Kolmogorov Complexity of Sets). Let A be a finite set of integers. Index A by a_i for i from 0 through $|A|$, where a_1 is the smallest value in A, a_2 is the next smallest, etc. Define a string representation of A to be the string $S =$ "$a_1; a_2; \ldots a_{|A|}$" with each a_i replaced by its literal value. Define $C(A)$ to be $C(S)$.

Let B be a finite set of pairs of integers. Index these pairs by $(a_{i1}; a_{i2})$ for i from 1 through $|B|$, where the a_is in dictionary order. Define a string representation of B to be the string $S =$ "$(a_{11}; a_{12})$\n $(a_{21}; a_{22})$\n $\ldots (a$n.c[(n)]TJ/F27 673iw312

Of course a more time-e cient sort is possible, but this Python program lets us more easily picture a corresponding Turing machine.

Appending programs comes at a length cost of log of the shorter program length, which is at most a constant for this program.

Running this sorting program after the shorter program selected by C_b gives us a program for S with the order a_i that has length $C_b + c$ for some constant c independent of S. Since C_a is de ned to be the length of the shortest program producing S with order a_i, C_a can't be larger than $C_b + c$. So we can see that printing any order of the elements of S is a problem in the same class of Kolmogorov complexity as C_a.

□

3.1.4 Kolmogorov Complexity versus Algorithm Complexity

There are many ways other ways of measuring how \hard" a problem is.

1. The usual metric of interest is time complexity: as the size of the input increases, roughly how many CPU cycles does the program take to run? The problem of factoring large primes is hard in this sense. While time complexity is de ned for speci c programs, it is also commonly used to describe the best known solution to a problem.

2. Another important metric is space complexity: as the size of the input increases, roughly how much active memory does the program require? Working with adjacency list representations of large matrices is hard in this sense.

3. The complexity of a particular program is sometimes described by how many branching points (conditional jumps) it has. High complexity in this sense indicates that a program is hard to maintain and debug.

4. Mathematicians and programmers are frequently interested in the di culty of coming up with any solution at all to a problem, informally measuring complexity by years left unsolved.

Kolmogorov complexity is independent of all of the above metrics (note that Kolmogorov complexity does not count memory used during computation, so it is not the same as space complexity).

Kolmogorov complexity is not frequently used in the eld of computer science, possibly because it is inconvenient. While any program gives an upper bound for the Kolmogorov complexity of the problem it solves, nding the true value is generally impossible. In addition, computers thankfully have enough memory these days that program size isn't much of a limitation. Finally, it seems to me that programmers simply refuse to write substantial programs which grow linearly with their inputs.

While Kolmogorov complexity doesn't have much in uence on concrete programs, it has value as an abstract measure of problem complexity.

De nition 3.9 (Computable Numbers). Computable numbers are real numbers which a Turing machine can approximate to any desired precision. All rational numbers and some irrationals, such as e, are computable. However, the computable numbers are countable because the set of Turing machines is countable, so most real numbers are uncomputable.

Because I am comparing Kolmogorov complexity to distance set size, approximations of real numbers are not precise enough for my purposes. I want to consider only numbers which are precisely nitely representable. These include the natural numbers and numbers $\frac{k}{2^l}$ for $k; l \in N$ because their

These two representations of numbers have analogs in actual computers, which store numbers in either of the forms $k \cdot 2^l : k, l \in \mathbb{Z}$ or $\frac{k}{m}$ for a fixed large natural m to provide both a large range of values alongside a good density of small numbers.

4 An Intuitive Relationship?

-3l712d05622/F28 9.2106 Tf[28.9l9627F0-9383 725.344 3 712(pro)28T]TJ/28omshortestJ/285m72(gramJ/28omtoJ/28o

Proof. If $n = 5$, the program is 55 characters long, which translates to 55 bytes or 440 bits. The program length increases by one character (or one byte or eight bits) for each extra digit in n, so the length of the whole program is at least the number of digits in n. The rest of the program runs correctly without modification for any n, so the remaining 54 bits of the program are constant. So, the length of this program for arbitrary n is $54 + \log_{10} n$. The Kolmogorov complexity of printing S is at worst this length: $C(S) \leq 54 + \log_{10} n \in O(\log n)$. □

5.2 No upper bound on Kolmogorov complexity from distance set size

Let s be an incompressible string of n bits. Divide s up into $\sqrt{n}$ segments: s_1 is the first $\sqrt{n}$ bits, s_2 is the next $\sqrt{n}$ bits, etc.

For each string s_i, define a set $S_i \in \mathbb{N} \cap [1, \sqrt{n}]$ such that k is in S_i exactly if the kth bit in s_i is 1. Then take $S = \bigcup \{S_i \times \{i\}\}$. S is an incompressible random subset of the integer grid $(\mathbb{N} \cap [1, \sqrt{n}])^2$, since if S was compressible we could reverse this construction and compress s. So any program for S requires at least n bits. This is much larger than the $\log n$ bits required to encode the complete $\sqrt{n} \times \sqrt{n}$ integer grid.

On the other hand, S is a subset of the $\sqrt{n} \times \sqrt{n}$ integer grid, Erdős' original bound says that S has at most $O(\frac{n}{\sqrt{\log n}})$ distances.

6 Sets of arbitrary Kolmogorov complexity and distance set size exist

In this section I will construct sets with minimal distance sets and arbitrary complexity, and sets with small complexity but maximal distance sets.

6.1 Sets of arbitrary complexity exist with $|\Delta| \in O(n)$

There is a program which prints $(0, 0)$ for all but m points, and either $(1, 0)$ or $(0, 0)$ at random for the remaining m points. This program has a minimum program length of $O(\log m)$ bits.

For n points, suppose we want an arrangement which has complexity $O(m)$ with $0 \leq m \leq n$. Place all n points along the x-axis. Assign the first $\lfloor m \rfloor$ points each the y-coordinate 0 or 1 at random, and assign all the remaining points the y-coordinate 0. Since the first m points have random y-coordinates which take 1 bit each to describe, any program returning this set must include at least n bits.

Here is a Python program which prints this set:

```
rand = [b1, b2, ..., bm]
for b in rand:
    print(f"({b},0)")
for i in range(n-m):
    print("(0,0)")
```

For each of the random coordinates b_i, decide arbitrarily whether bi in the program is literally 1 or 0.

Since most strings of any given length are incompressible, we can choose rand to be incompressible, which means there is no more efficient way to remember which points are shifted. Then we can't avoid spending at least m bits.

Surprisingly, this set has very few distances. All points lie on an integer grid between $(0, 0)$ and $(n, 1)$, so its distance set is a subset of this grid section's distance set. The possible distances are those between points with $x = 1$, those between points with $x = 0$, and those between a point with $x = 1$ and a point with $x = 0$:

its derivative with respect to m is

From the other direction, Kolmogorov complexity is closely related to Hausdor dimension, and a few researchers have developed a constructive Hausdor dimension which might serve as a more useful description of complexity [4].

Either an energy-based metric or constructive Hausdor dimension seem likely to work for in nite sequences and real numbers.

10 Acknowledgements

Thank you to my advisor Professor Iosevich for introducing me to this problem and guiding me through the research process, as well as for his constant encouragement throughout my undergraduate career. I was incredibly lucky to nd such an excellent teacher and mentor.

Thanks also to James Iler and Ben Coukos-Wiley for pointing out many errors, Alex Prideaux for