Kolmogorov Complexity and Distance Sets: Two Notions of Set Complexity

Elana Elman

April 26, 2024

Abstract

I introduce Kolmogorov complexity and the Erdös 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 finite sets of fixed 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 finite 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 finite string; in the context of sets, I consider the shortest program which lists the coordinates of all the set's points. The distance set size, taken from Erdös' distinct distances problem, counts the number of unique distances between points in a set. Each of these intuitively seems to measure how regular a set is. However, I find that they are unrelated concepts: knowing a set's Kolmogorov complexity gives no information about its distance set size, and vice versa.

2 The Erdös Distinct Distance Problem

Definition 2.1 ($\Delta(P)$)**.** For a set $S \in \mathbb{R}^2$, the distance set $\Delta(P)$ is the set of unique distances between points in P.

Note that $|\Delta(P)|$ is the number of distinct distances between points in P.

Erdös' distinct distance problem asks what the smallest possible number of distinct distances in any arrangement of n points is. For example, three points may be arranged into a regular triangle, producing one unique distance, but four points can at best be arranged into a square, producing two unique distances (the square's side length and its diagonal length).

Let $\mathbf{P_n}$ be the collection of all sets of n different points in $\mathbb{R}^2$.

Definition 2.2 ($M(n)$)**.** For a positive integer n, let $M(n)$ be the smallest possible number of distinct distances in any arrangement of n different points on the plane:

$$M(n) = \min_{P \in \mathbf{P_n}} |\Delta(P)|$$

The precise value of $M(n)$ is only known for very small n, and work on the distinct distance problem seeks to bound M asymptotically:

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

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

Erdös' 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 $|\Delta(P)| = \lfloor \frac{n}{2} \rfloor \in O(n)$.

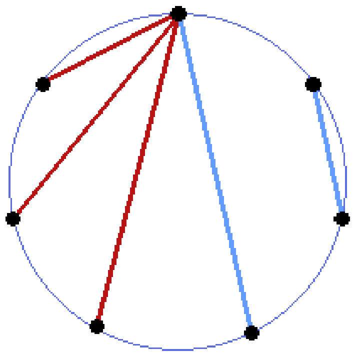

Figure 1: 7 points on a circle. One instance of each of this set's distances is drawn in red. The blue lines are examples of repeated distances.

Example (Erdös' upper bound)**.** For the sake of simplicity, assume n is a square integer. Let P be n points arranged on a $\sqrt{n} \times \sqrt{n}$ integer grid, with each coordinate running from 0 through $\sqrt{n} - 1$. Then

$$|\Delta(P)| = \left|\left\{\sqrt{(x_{p_i} - x_{p_j})^2 + (y_{p_i} - y_{p_j})^2}\right\}\right| = \left|\left\{(x_{p_i} - x_{p_j})^2 + (y_{p_i} - y_{p_j})^2\right\}\right|$$

$$\leq \left|\left\{a^2 + b^2 | a, b \in \mathbb{Z}_+ \cap [0, \sqrt{n} - 1]\right\}\right| \leq \left|\left\{a^2 + b^2 | a, b \in \mathbb{Z}_+, a^2 + b^2 \leq 2n\right\}\right| \leq \frac{bn}{\sqrt{\log n}} \in O(\frac{n}{\sqrt{\log n}})$$

The last inequality is a limit by Landau.

Figure 2 shows a 6×6 integer grid and illustrates why the number of distinct distances is $< \frac{n}{2}$:

- Any distance between points on the grid can be rotated and shifted to start at the point highlighted in blue, so the set of distinct distances in the grid is the number of distinct distances from the blue point to other points. There are $n - 1$ other points in the grid, so $|\Delta| \leq n - 1$.
- Any point outside the green triangle is the same distance from the blue point as its diagonally opposite point, so counting distances from the blue point to points in the green triangle is sufficient. There are $\sum_{i=1}^{\sqrt{n}} i = \frac{\sqrt{n}(\sqrt{n}-1)}{2} = \frac{n-\sqrt{n}}{2}$ distances from the blue point to points in the green triangle, so $|\Delta| < \frac{n}{2}$.

- 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

$$x = k(2ab)$$
$$y = k(a^2 - b^2)$$
$$h = k(a^2 + b^2)$$

  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 $O(n - \sqrt{n}\log n)$.

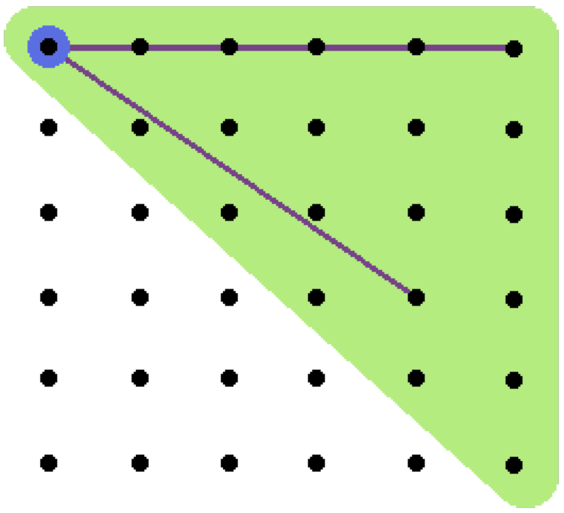

Figure 2: A 6×6 square grid. All distances outside the green triangle are also present in the green triangle. The green triangle contains some repeating distances: two lines of length 5 are drawn in purple.

2.0.2 Lower Bounds

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

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

[1]For justification, please attend Fuyi Kuang's thesis defense, which includes a calculation of the number of lattice points under a hyperbola.

points, so some hemisphere has at least $\frac{\sqrt{n}}{2}$ points. The distance from the leftmost point on this hemisphere to each other point on the hemisphere is different, so there are at least $\frac{\sqrt{n}}{2} \in O(\sqrt{n})$ distances in P.

- Otherwise $m >= \sqrt{n}$, and each circle represents a distance from a to another point of p, so there are at least $\frac{\sqrt{n}}{2} \in O(\sqrt{n})$ distances in P.

□

3 Kolmogorov Complexity

Kolmogorov complexity is an concept from information theory which is designed to measure randomness. Randomness is usually a property ascribed to generating processes, not individual objects, but Kolmogorov complexity allows us to argue that the string "000000000000" is less random than "128683487345". The Kolmogorov complexity of a string is the length of the shortest program that outputs it. In this situation, I will precisely define "programs" and describe properties of Kolmogorov complexity.

3.1 Formalizing Computation

3.1.1 Turing Machines

Definition 3.1 (Turing machines)**.** A Turing machine is a formalization of computing. It consists of a finite set of states with a finite collection of transitions between states which accesses an infinite tape through a pointer. On transitions, the machine may overwrite the currently accessible tape cell with a new symbol and/or move the pointer right or left. The Turing machine must also specify a set of acceptable states to end in and an input string composed of finitely many non-blank characters from a finite alphabet which is written on the tape.

Any Turing machine may be finitely described by a tuple $\langle \Sigma, b, S, s_0, \delta, F \rangle$, where Σ is the alphabet of the input string, $b \in \Sigma$ is the blank character, S is the set of states, s_0 is the starting state, δ is the transition function $(S - F) \times \Sigma \to S \times \Sigma \times \{L, R, N\}$, and F is any subset of S. δ may be a partial function, in which case a global state with no successor halts.

Example (Increment)**.** Let $\Sigma = \{0, 1, B\}$, $b = B$,
$S = \{\texttt{SEEK}, \texttt{CARRY}, \texttt{DONE}\}$, $s_0 = \texttt{SEEK}$, $F = \{\texttt{DONE}\}$, and δ is the function

(state, symbol)	next state	write	move
(`SEEK`, 0)	`SEEK`	0	R
(`SEEK`, 1)	`SEEK`	1	R
(`SEEK`, B)	`CARRY`	B	L
(`CARRY`, 0)	`DONE`	1	N
(`CARRY`, 1)	`CARRY`	0	L
(`CARRY`, B)	`DONE`	1	N

Intuitively, this Turing machine moves to the right side of the input (`SEEK`), then flips bits from right to left until it's able to put a 1 in a previously-0 cell (`CARRY`). We can now (inefficiently) add n to positive integers by writing n 1s to the left of our input and erasing one per run of this increment routine. Positive integer addition can be (extremely inefficiently) built into integer multiplication, and so on; this capacity for arithmetic plus the recursive structure provided by state transitions allows for arbitrary computation. Notably, there exists a Turing machine which takes as input descriptions of another Turing machines and simulates their execution, returning the result the specified Turing machine would return.

Time	Tape	Current State	Next State	Write	Move
0	...$B\underline{1}01B$...	`SEEK`	`SEEK`	1	R
1	...$B1\underline{0}1B$...	`SEEK`	`SEEK`	0	R
2	...$B10\underline{1}B$...	`SEEK`	`SEEK`	1	R
3	...$B101\underline{B}$...	`SEEK`	`CARRY`	B	L
4	...$B10\underline{1}B$...	`CARRY`	`CARRY`	0	L
5	...$B1\underline{0}0B$...	`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 finite memory. Turing machines are also difficult 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 finite, there exist finite translations between Turing-equivalent languages. This shows that finite compilers – programs that translate between languages – exist. We may run a program in a different language by appending a compiler to the program. This adds a constant length to the program which depends on the two languages in use.

For the rest of this paper I will write programs in Python, but they could be written in any other language with only a constant penalty to length.

3.1.3 Defining Kolmogorov Complexity

Definition 3.2. The length $l(p)$ of a natural number p is the number of digits in its binary representation.

Definition 3.3 (Kolmogorov complexity with respect to a specifying function)**.** Let S be a countable set and enumerate it with natural numbers $n(x)$. The Kolmogorov complexity of $x \in S$ with respect to the partial function f on natural numbers is

$$C_f(x) = \min_{p \in \mathbb{N}}(\{l(p) | f(p) = n(x)\} \cup \{\infty\}).$$

In terms of computing, S is a set of outputs, f is a programming language, p are programs, and C_f reports the length of the shortest program that returns x when interpreted in language f. C_f returns infinity if no program in f returns x.

Definition 3.4 (Minorizing functions)**.** A function f minorizes a function g if there is a constant c such that $C_f(x) \leq C_g(x) + c \ \forall x \in S$.

Definition 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 difference 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.

Definition 3.6 (Computable Functions)**.** A partial function $f : \mathbb{N} \to \mathbb{N}$ is called computable if there is some Turing machine which terminates with output $f(n)$ on each input n for which f is defined.

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 $l(g)$ `1`s, then one `0`, then the literal description g of the desired Turing machine, then the literal input string p. U may then use the prefix of `1`s to separate g from p and run its simulation. Let f_U be the function executed by U. □

Definition 3.7 (Kolmogorov Complexity)**.** The Kolmogorov complexity of a finite string s is defined 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} different 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 finite 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_C longer than `K`: `C` is $a + c_C$ bits long.

And write a third function `B` which returns a string with arbitrarily high Kolmogorov complexity:

```
def B(min):
    i = 1
    while true:
        if i > min and not C(i):
            return i
        i += 1
```

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 Vitányi'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 Erdös 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`$\ldots (a_{n1}, a_{n2})$" where each (a_{i1}, a_{i2}) is replaced by its literal values. Define $C(B)$ to be $C(S)$.

Lemma. Most subsets of $\mathbb{N} \cup [0, \sqrt{n})$ are incompressible.

Proof. Given a subset S of $\mathbb{N} \cup [0, \sqrt{n})$, consider the string `s` with a length of $\sqrt{n}$ bits, where the ith bit is `1` if $i \in S$ and `0` otherwise. This relationship is bijective between subsets S and $\sqrt{n}$-length strings.

If S is compressible, let the program P have length $l_P < \sqrt{n}$ and output S. Construct a new program P' which generates S and then converts it to its bit-array representation as above; this algorithm adds only a small constant number of bits. Now P' is an algorithm with length $l_P + c < \sqrt{n}$ for sufficiently large n which generates S's bit-array representation.

Since we can do this for each compressible subset S and generate a unique string of length $\sqrt{n}$, s is compressible if S is. But most $\sqrt{n}$-length strings are incompressible, so most subsets S must be also.

□

Theorem 4. The Kolmogorov complexity of a set S does not depend on the order in which the elemts of S are returned.

Proof. Suppose that any set $S \in \mathbb{R}^2$ with $|S| = n$ has an order b_i such that $C(S)$ with respect to b_i (call this value C_b) is less than $C(S)$ with respect to a_i (call this value C_a).

If a list of numbers is already in the computer's memory at `list1`, sorting it is possible with a constant-length addition to the program:

```
list2 = []
while list1:  #this means "while list1 is not empty:"
    minimum = list1[0]
    for k in list1:
        if (k[0] < minimum[0] or
           (k[0] = minimum[0] and k[1] < minimum[1])):
            minimum = k
    list1.remove(k)
    list2.append(k)
```

Of course a more time-efficient 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 defined 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.

$\square$

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 defined for specific 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 difficulty 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 field of computer science, possibly because it is inconvenient. While any program gives an upper bound for the Kolmogorov complexity of the problem it solves, finding 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 influence on concrete programs, it has value as an abstract measure of problem complexity.

Definition 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 finitely representable. These include the natural numbers and numbers $\frac{k}{2^l}$ for $k, l \in \mathbb{N}$ because their binary representations are finite. Any other rational number may be represented as fractions by delimiting pairs of integers with a new tape symbol, since we can perform arithmetic operations perfectly well on fractions.

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?

Kolmogorov complexity and the Erdös distinct distance problem have a number of commonalities:

- For a given set, Kolmogorov complexity and the size of the distance set give a sense of how constrained the set is.
- The shortest program to output a set is incomputable and the arrangement producing the fewest distances is extremely difficult to find.
- Both Kolmogorov complexity and distance set size are extremely sensitive to slight variation – any degree of randomness added to any set brings both its Kolmogorov complexity and its distance set size up to the maximum.

After noticing these similarities, I wondered if a formal direct relationship exists between these topics. In the rest of this paper, I will show that these concepts are not directly related.

5 Kolmogorov Complexity and Distance Set Size are Independent

I will show that Kolmogorov complexity and distance set size are not directly related by constructing sets of minimal complexity with many distances, and sets of large complexity with minimal distances.

5.1 No upper bound on distance set size from Kolmogorov Complexity

Given an integer n, consider the following Python program:

```
for i in range(0, n):
  x = 2**i
  print(f"({x}, 0)\n")
```

After substituting in the actual integer for `n`, this program prints a list of n points $(2^m, 0)$ for $m \in \mathbb{N}, m < n$.

The set $S = \{(2^m, 0) | m \in \mathbb{N}, m < n\}$ has no duplicate distances between its points:

Proof. Of course a set of one point has no duplicate distances.

Suppose the set $S^{m-1} = \{(2^k, 0) | k \in \mathbb{N}, k < m - 1\}$ has no duplicate distances. $(2^m, 0)$ is 2^{m-1} away from $(2^m - 1, 0)$ and further from each other point in S^{m-1}. Since the maximum distance within S^{m-1} is $2^{m-1} - 1$, all distances between $(2^m, 0)$ and points in S^{m-1} are greater than all distances within S^{m-1}. Also, $(2^m, 0)$ is a distinct distance $2^m - 2^k, k < m$ from each point in S^{m-1}. The distances in $S^m = \{(2^k, 0) | k \in \mathbb{N}, k < m\}$ are the distances in S^{m-1} and the distances between $(2^m, 0)$ and points in S^{m-1}, and since there are no repeated distances within or between these groups, S^m has no duplicate distances. $\square$

So S has the maximum possible number of distances in any set of n points in $\mathbb{R}^2$, which is $\binom{n}{2} \in O(n^2)$.

On the other hand, the program above has the minimum possible length for any program containing the number n, so the problem of listing the points in S has very low Kolmogorov complexity:

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: $C(S) \leq 54 + log_{10}n \in O(\log n)$. $\square$

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 $S = \cup\{S_i \times \{i\}\}$. 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, Erdös' 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 $x = 0$:

- Between points which both have $x = 0$, only integer distances $\leq n$ are possible. The same is true between points which each have $x = 1$. Each of these cases produce at most n distinct distances.
- Any distance between a point with $x = 0$ and a point with $x = 1$ is the same as the distance from $(0, 0)$ to some point with $x = 1$, so this case also produces at most n distinct distances.

So $|\Delta|$ here is at most $3n \in O(n)$.

6.1.1 Sets of arbitrary complexity exist with $|\Delta| \in O(\frac{n}{\sqrt{\log n}})$

Place n points on an $\sqrt{n} \times \sqrt{n}$ grid. Then decide arbitrarily whether to shift each of the first m points right by $\sqrt{n}$. The total number of distances in this arrangement is no more than the number of distances in a $2\sqrt{n} \times 2\sqrt{n}$ grid which has $4n$ points and $O(\frac{4n}{\sqrt{\log 4n}}) \leq O(\frac{4n}{\sqrt{\log n}}) = O(\frac{n}{\sqrt{\log n}})$ distances.

A program for this set is

```
import math
rand = [t1, t2, ..., tm]
s = math.sqrt(n)

index = 0
for i in range(1, s):
    for j in range(1, s):
            if index < m:
                if t[index]:
                    print(f"({i+s},{j})")
                else:
                    print(f"({i},{j})")
                index += 1
            else:
                print(f"({i},{j})")
```

Where each `ti` is a boolean representing whether its corresponding point is shifted.

As in the linear case, most bit strings of length m are incompressible, so some sequence of `tis` gives a set with a complexity of at least m bits. We can also see from this program that every sequence of `tis` gives a set with a complexity of at most $O(m)$, since the whole program is constant with respect to m except for the length of `rand`.

6.2 Sets with arbitrary $|\Delta|$ exist with complexity $O(\log n)$

For any $n \in \mathbb{N}$, suppose we want an arrangement of n points with $O(n^k)$ distances for $k \in [1, 2]$.

Let $M \subset \mathbb{R}^2$ be an even number m of the n points evenly-spaced around a circle of radius $\frac{1}{4}$ centered at the origin, with one of the points at $(0, \frac{1}{4}$, so that there are $\frac{m}{2}$ distinct distances in M. Let $K \subset \mathbb{R}^2$ be the remaining $n - m$ points placed at $(2^i, 0)$ for $i \in \mathbb{N}, i < m - n$, so that there are $\binom{n-m}{2}$ distinct distances in K.

How many distinct distances exist between points in M and points in K?

For a fixed point k in K and a point m_0 in M, note that m_0 sits on a circle of radius $\frac{1}{4}$ around the origin and a circle of some radius d around k. If some other point in M also has distance d from k, it must also be on both of these circles. Since the two circles intersect at at most two points, one of which is m_0, at most one other point in M is d away from k. In fact, since k is on the x-axis, the other point in M at which is $|k - (x, y)$ away from k is $(x, -y)$. Since M is by design symmetrical across the x-axis, the number of distances from k to M is exactly $\frac{m}{2}$.

Now consider any $k' \in K$ with $k \neq k'$. Without loss of generality, suppose k falls to the left of k'. I will use the following calculation to show that k and k' do not share any distances to points in M:

Lemma. For any $a \in M$ and $k_0 = (2^i, 0) \in K, |a - k_0| \in [2^i - \frac{1}{4}, 2^i + \frac{1}{4}]$:

Proof.

$$a \in M \to a_y = \pm\sqrt{\frac{1}{4^2} - a_x^2}, a_x \in \Big[-\frac{1}{4}, \frac{1}{4}\Big]$$

$$\to |k_0 - a| = \sqrt{(2^i - a_x)^2 + (0 - a_y)^2} = \sqrt{(2^i - a_x)^2 + \Big(\pm\sqrt{\frac{1}{4^2} - a_x^2}\Big)^2}$$

$$= \sqrt{(2^i)^2 - 2a_x 2^i + a_x^2 + \frac{1}{4^2} - a_x^2} = \sqrt{(2^i)^2 - 2a_x 2^i + \frac{1}{4^2}}$$

$$\in \left[\sqrt{(2^i)^2 - 2 \cdot \frac{1}{4} \cdot 2^i + \frac{1}{4^2}}, \sqrt{(2^i)^2 + 2 \cdot \frac{1}{4} \cdot 2^i + \frac{1}{4^2}}\right]$$

$$= \left[\sqrt{(2^i - \frac{1}{4})^2}, \sqrt{(2^i + \frac{1}{4})^2}\right] = \left[2^i - \frac{1}{4}, 2^i + \frac{1}{4}\right]$$

□

Since $k < k'$ and both are $\in K$, $k = (2^i, 0)$ and $k' = (2^j, 0)$ for $0 \leq i < j$, so $|k_1 - k_0| \geq 1$. Based on this and the lemma above, $|k - a| \leq 2^i + \frac{1}{4} \forall a \in M$ and $\forall b \in M, |k' - b| \geq 2^j - \frac{1}{4} > 2^i + \frac{3}{4} > 2^i + \frac{1}{4} \geq |k - a|$, so $|k - a| \neq |k' - b|$.

So no distances between M and one k are duplicated with another k' in K.Now there are $\frac{m}{2}$ distances from a specific k to points in M, and these distances are different for each of the $n - m$ points in K, so the total number of distances between M and K is $(n - m)\frac{m}{2}$.

Are any of these distances between M and K shared with distances within M or within K? No, since every distance within M is $\leq \frac{1}{2}$; every distance between M and K is $\geq \frac{3}{4}$ and of the form $\sqrt{2^i + \frac{1}{16}}$ so not an integer, and every distance within K is an integer.

Now the total number of distinct distances in $M \cup K$ is

$$\frac{m}{2} + \binom{n-m}{2} + (n - m)\frac{m}{2}$$

$$= \frac{m}{2} + \frac{(n-m)(n-m-1)}{2} + \frac{nm - m^2}{2}$$

$$= \frac{n^2 - nm - n + 2m}{2}$$

for $m \leq n$. To get n^α distances, set $m = \frac{n^2 - n^\alpha}{n-2}$; note that m exists and is nonnegative for $n > 2$, and since $\alpha \geq 1$ it is less than n. Now

$$|\Delta| = \frac{n^2 - n(\frac{n^2 - n^\alpha}{n-2}) - n + 2(\frac{n^2 - n^\alpha}{n-2})}{2} = \frac{n^2 - (\frac{n^2 - n^\alpha}{n-2})(n-2) - n}{2}$$

$$= \frac{n^2 - (n^2 - n^\alpha) - n}{2} = \frac{n^\alpha - n}{2} \in O(n^\alpha)$$

since $\alpha > 1$.

Unfortunately, m according to this formula might not be an integer, much less even. An arbitrary real number m_0 is no more than 2 below an even number; let m_1 be the smallest even number $\geq m_0$, so that $m_0 \leq m_1 \leq m_0 + 2$. Since $|\Delta| = \frac{n^2 - nm - n + 2m}{2}$ is a continuous function on real numbers and its derivative with respect to m is $-\frac{n}{2}+1$, $|\Delta|$ strictly decreases as m increases whenever $n > 2$, so the number of distinct distances for $m = m_1$ is between $|\Delta|$ for m_0 and $|\Delta|$ for $m_0 + 2$.

So,

$$\frac{n^2 - n(\frac{n^2-n^\alpha}{n-2}) - n + 2(\frac{n^2-n^\alpha}{n-2}))}{2} \leq |\Delta_{m_1}| \leq \frac{n^2 - n(\frac{n^2-n^\alpha}{n-2} + 2) - n + 2(\frac{n^2-n^\alpha}{n-2} + 2)}{2}$$

$$\rightarrow \frac{n^\alpha - n}{2} \leq |\Delta_{m_1}| \leq \frac{n^2 - n(\frac{n^2-n^\alpha}{n-2}) - 2n - n + 2(\frac{n^2-n^\alpha}{n-2}) + 4}{2} = \frac{n^\alpha - 3n + 4}{2}$$

$$\rightarrow |\Delta_{m_1}| \in O(n^\alpha)$$

This shows that for any α between 1 and 2, some integer $m < n$ gives $|\Delta| \in O(n^\alpha)$.

7 Arguments Against Kolmogorov Complexity

As I explored a potential connection between distance sets and Kolmogorov complexity, I discovered that Kolmogorov complexity lacks many helpful properties for describing set complexity.

- Kolmogorov complexity is not able to describe most sets in $\mathbb{R}^2$ because most numbers are not finitely representable.
- Kolmogorov complexity does not increase with added points: a subset may have drastically higher complexity than its superset.
- Kolmogorov complexity doesn't capture symmetry, which would be preferable when discussing geometric structure. The set of points 2^i for $0 \leq i < n$ has the same Kolmogorov complexity as the set of points i for $0 \leq i < n$.

8 An Argument for Kolmogorov Complexity

Kolmogorov complexity doesn't help much with any computation; time complexity is much more useful to describe the cost of running any algorithm, and the space taken up by a program is usually trivial in comparison to memory needed to run the program, much less the total memory of a modern computer. Kolmogorov complexity seems to actually be for people: an algorithm with short Kolmogorov complexity has fewer steps, and possibly fewer logical leaps, to get through a human head.

It is logically simple (though not physically feasible) to find a shortest program for a specific string S given its Kolmogorov complexity. If it were even sometimes possible to find an upper bound for a set's complexity and translate this bound into a shorter program, we would sometimes be able to make these sets more easily understandable. Extending this possibility to programs in general, such as neural networks, highlights the possible power of such a development.

9 Alternatives for Set Complexity

I considered space complexity as an alternative to Kolmogorov complexity on the intuition that Kolmogorov complexity "cheats" by hiding work in computations, but I found that space complexity doesn't improve any of the examples in this paper.

At the start of my thesis, I had strong intuition that Kolmogorov complexity of a set is analogous to the energy of a physical structure, such as a crystal structure for a regular set. I found that this link between small Kolmogorov complexities and regular structures doesn't exist, but I expect that a related measure of energy might capture the intuitive link between distance set size and structuredness.

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

Either an energy-based metric or constructive Hausdorff dimension seem likely to work for infinite 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 find such an excellent teacher and mentor.

Thanks also to James Iler and Ben Coukos-Wiley for pointing out many errors, Alex Prideaux for the Pythagorean triangles, and all of the above plus Fuyi Kuang, Lyrica Yanaway, and Baby the cat for their friendship and support during this project.

References

[1] P. Erdos. "On Sets of Distances of n Points". In: *The American Mathematical Monthly* 53.5 (1946), pp. 248–250. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2305092 (visited on 04/15/2024).

[2] Jerzy Kocik. "Clifford Algebras and Euclid's Parameterization of Pythagorean Triples". In: *Advances in Applied Clifford Algebras* (2006). DOI: https://doi.org/10.1007/s00006-006-0019-2.

[3] Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. 3rd ed. Springer Publishing Company, Incorporated, 2008. ISBN: 0387339981.

[4] Elvira Mayordomo. "A Kolmogorov complexity characterization of constructive Hausdorff dimension". In: *Information Processing Letters* 84.1 (2002), pp. 1–3. ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(02)00343-5. URL: https://www.sciencedirect.com/science/article/pii/S0020019002003435.

[5] Peter Bro Miltersen. *Course notes for Data Compression - 2 Kolmogorov complexity*. 2005. URL: https://web.archive.org/web/20090909132048/http://www.daimi.au.dk/~bromille/DC05/Kolmogorov.pdf.