The Complexity of Compression

Rahul Santhanam

University of Oxford

Plan of Talk

- Motivation
- Intro to Kolmogorov Complexity
- Resource-Bounded Variants
- Learning
- Cryptography

Plan of Talk

- *Motivation*
- Intro to Kolmogorov Complexity
- Resource-Bounded Variants
- Learning
- Cryptography

Data Compression

- Data compression is a fundamental task in computer science
  - Communication: When sending data across a communication channel, we would like it to be as *compact* as possible to save on time and space costs
  - Learning: At a high level, the main goal in learning is to find a *compact* hypothesis that explains the training data and performs well on the test data
  - Cryptography: Encryption relies on the ability to efficiently produce pseudorandom strings that are indistinguishable from random strings, though pseudorandom strings are *compressible* in principle and random strings are not
- Theoretical foundations for data compression
  - How do we characterize the inherent compressibility of a dataset?
  - Is there an efficient procedure to optimally compress a dataset?

Shannon's Theory

- Shannon's theory of source coding provides good answers to these questions when we are dealing with *distributions* on data
  - In this case, we know that the *entropy* of a distribution is the optimal expected compression length, and the efficient Huffman coding procedure achieves this
  - Useful if there is a reasonable way to model distributions, eg., the Zipf law on natural language utterances
  - But what if we have no prior knowledge of this form, and we are interested in the *inherent* compressibility of data (modelled simply as a finite string)?

Inherent Compressibility

- It is clear that some strings should be much compressible than others, eg., a string of N zeroes should be more compressible than a random string
- Explicit redundancies in strings, such as re-occurring patterns, are exploited in algorithms such as the Lempel-Ziv algorithm and its variants
- But there could be redundancy that is not based on repetition
  - Eg., consider the strings “3141592653” and “2718281828”
  - Anyone with a basic knowledge of calculus can see that these are easily distinguishable from random 10-digit strings ☺

Plan of Talk

- Motivation
- *Intro to Kolmogorov Complexity*
- Resource-Bounded Variants
- Learning
- Cryptography

Foundations of Compressibility

- A compression scheme over an alphabet Σ is a pair of functions C, D: $\Sigma^* \rightarrow \Sigma^*$ such that for all x in Σ^*, D(C(x)) = x, and ideally
  - The compressor C is close to *optimal* in that |C(x)| is not much larger than |C'(x)| for *every* x and *every* compression scheme C'
  - The de-compressor D is (efficiently) computable
  - C is (efficiently) computable
- These criteria are in tension with each other. For now we prioritize optimality, and define a measure called *Kolmogorov complexity* which captures the *inherent compressibility* of a string

Defining Kolmogorov Complexity

- Let U be a fixed universal Turing machine
- For any string x in Σ^*, $K(x)$ is min $\{|p|: U(p, \varepsilon) = x\}$
- Intuitively, $K(x)$ is the size of the smallest program that produces x when run on the empty string
- Examples
  - $K(0^N) \leq \log(N) + O(1)$, since we can describe 0^N (in a way that makes sense to a computable de-compressor) by using $\log(N)$ bits to describe N and $O(1)$ bits to describe a program that outputs 0^N given N
  - $K(\pi_N) \leq \log(N) + O(1)$, where π_N is the string consisting of the first N bits of π

Basic Properties

- (1) For every x in Σ^*, $K(x) \leq |x| + O(1)$
  - Any string x can be described by itself together with a program p of constant size that just prints x out
- (2) For each integer n, there is x of length n such that $K(x) \geq n$
  - Straightforward counting argument
  - For any i, there are at most 2^i strings of Kolmogorov complexity i (since there are at most 2^i descriptions of length i)
  - So there are at most 2^n-1 strings of Kolmogorov complexity $< n$
  - By pigeonhole principle, there is a string x of length n with $K(x) \geq n$

Near-Optimality

- For any compression scheme with compressor C and computable de-compressor D, for every string x, $K(x) \leq C(x) + O(1)$
- The reason is simple: since D is computable, there is some program p of size $O(1)$ that computes it. Hence every x can be described by $C(x)$ together with p
- Kolmogorov complexity has a very simple definition but very strong properties!

Is Kolmogorov Complexity Computable?

- Kolmogorov complexity corresponds to a compression scheme where the de-compressor is implemented by a universal Turing machine U
- Nice property: The maximum to which we can compress any string x is roughly $K(x)$
- However the following fundamental question about the compression scheme remains: given a string x, can we compute how much we can compress it?
- Answer, sadly, is no! But the proof is very elegant, and is a version of *Berry’s Paradox*

Berry’s Paradox

- Consider the expression “The smallest positive integer not definable in under sixty letters”
- This expression has 57 letters, so if “definability” has a clear meaning, we get a contradiction
  - Let N be the value of the expression
  - We have that N cannot be defined in under 60 letters
  - However, we have just given an expression with 57 letters that defines it!
- We are led to the conclusion that “definability” cannot have a clear meaning when considering expressions such as the above
- An argument of a very similar flavour can be applied to Kolmogorov complexity

Uncomputability of Kolmogorov Complexity

- Suppose, for the sake of contradiction, that there is a TM M that computes K
- Define a TM N that accepts x iff $K(x) \geq n$
  - By Basic Property (2) of K complexity, N accepts at least one string for each input length n
- Now define a sequence of strings $\{x_n\}$, $|x_n|=n$, as follows
  - For each n, x_n is the lexicographically first string of length n that N accepts
  - Note that we can compute x_n given n by simulating N on strings of length n in lex order and outputting the first such string it accepts
  - This implies that $K(x_n) \leq \log(n) + O(1)$
  - But, by definition of x_n, $K(x_n) \geq n$ for each n, which is a contradiction for large enough n

From Computation to Proofs

- These seemingly elementary considerations about Kolmogorov complexity point to deep issues in the foundations of mathematics!
- Recall Godel's First Incompleteness Theorem: No consistent effectively axiomatizable proof system can prove all truths about the arithmetic of natural numbers
- We can get strong incompleteness results by arguing about Kolmogorov complexity in a similar way to how we showed uncomputability

The Deep Intractability of Kolmogorov Complexity

- Theorem [Chaitin]: Let X be any effectively axiomatizable sound proof system. There are only finitely many statements of the form "$K(x) \geq m$" that can be proved in X!
- Proof: Suppose, for the sake of contradiction, that there are infinitely many statements of the form "$K(x) \geq m$" that are provable in X. This implies that there are infinitely many m for which some statement "$K(x) \geq m$" is provable in X. Given m, we can computably find an x such that "$K(x) \geq m$" is provable in X by enumerating potential proofs of such statements in parallel until we find an actual one. But this x has $K(x) \leq \log(m) + O(1)$, and for large enough m, this contradicts $K(x) \geq m$ (which is implied by the soundness of X)

Takeaways

- Kolmogorov complexity provides a natural measure of “inherent compressibility of a string”
- However, Kolmogorov complexity is not computable, and as a consequence, the corresponding compression scheme does not have an efficient compressor
- Kolmogorov complexity seems on the surface just to be an elementary concept about data compression, but it leads to deep insights into the foundations of mathematics

Plan of Talk

- Motivation
- Intro to Kolmogorov Complexity
- *Resource-Bounded Variants*
- Learning
- Cryptography

Kolmogorov Complexity with Resource Bounds

- Kolmogorov complexity is not very usable in practice for data compression
  - The de-compressor has no a priori time bound
  - The compressor is not even computable!
- We consider versions where the de-compressor is more efficient
- Given polynomial time bound t, let $K^t(x) = \min\{\, |p| : U(p) = x$ in at most $t(|x|)$ steps$\}$
- Note that de-compressor now runs in polynomial time in the size of the source data

Does Near-Optimality Still Hold?

- Nearly ☺
- Proposition: Suppose there is a compression scheme with compressor C and de-compressor D, where the de-compressor D runs in time t (as a function of the length of its output). Then for each x, $K^{O(t \log(t))}(x) \leq C(x) + O(1)$
- Proof: Exactly the same as the proof of the corresponding Proposition for standard Kolmogorov complexity, except that we now use a time-efficient universal TM that simulates a time t TM in time $O(t \log(t))$

The Complexity of Compression

- Time-bounded Kolmogorov complexity yields a “near-optimal” compression scheme with polynomial-time de-compression
- Key question: can compression be done in polynomial time? This would make the compression scheme ideal for use in a resource-bounded world
- For unbounded Kolmogorov complexity, we could *prove* that compression could not be done efficiently, or even computably
- However, this proof does not directly carry over to the resource-bounded setting

NP vs P

- Recall the NP vs P problem: is every computational problem where solutions are poly-time verifiable also poly-time solvable?
- This is the main question of theoretical computer science, and one of the 6 unsolved Clay Millennium Problems
- NP vs P turns out to be closely connected to the question of whether the K^t compression scheme has an efficient compressor!

The Connection

- If NP=P, then the K^t scheme does have an efficient compressor
  - Guess the smallest program p for which U(p) outputs x within t(|x|) steps
  - Verifying that p is the smallest program isn't obviously polynomial time, as we need to check if there *exists* a smaller program that works
  - However, the assumption that NP=P can be used to do this check in poly time! Then, by using the assumption again, we can *find* the smallest program in polynomial time
- However, most researchers believe that NP $\neq$ P. What then?
- This relates to a central open question about Kolmogorov complexity: is K^t NP-hard to compute? If so, then NP = P if and only if the K^t scheme has an efficient compressor!

Takeaways

- By defining resource-bounded versions of Kolmogorov complexity, we obtain compression schemes that satisfy a version of near-optimality while being having poly-time decompression algorithms
- Whether these compression schemes also have poly-time compression algorithms is closely related to the NP vs P problem
- Important open problem: Is computing K^t complexity NP-hard? If so, whether the corresponding compression scheme has poly-time compression is *equivalent* to NP ≠ P

Plan of Talk

- Motivation
- Intro to Kolmogorov Complexity
- Resource-Bounded Variants
- *Learning*
- Cryptography

The Problem of Induction

- Learning theory, as well as the process of doing science itself, are deeply concerned with the problem of induction: how to extrapolate a pattern based on limited observations?
- Ray Solomonoff showed how to solve the problem of induction in a *mathematically rigorous* way by using the tools of Kolmogorov complexity

The Setting

- We model observations in the most simple way possible – as a sequence of bits
- Given an observed sequence x, prediction corresponds to *extending* this sequence in a meaningful way
- Assumption: sequence is produced by some *computable* process that is deterministic
  - Justification: By the Universal Church-Turing thesis, processes that occur in Nature can be modelled as computable
- Challenge: There can be several different computable processes that are consistent with the observations but yet make different predictions

Occam's Razor

- The principle of Occam's Razor says that the *simplest* explanations are most likely
  - "Simple" = "has low Kolmogorov complexity"
  - Example: For the sequence 01010101010101, the most reasonable prediction for the next bit is 0, but the prediction of 1 should not be ruled out
- But how do we weight explanations according to their simplicity?
  - Use a *Bayesian* approach by defining the *universal prior* on observations as follows: to generate an observation sequence of length n, generate a program p with probability proportional to $2^{-|p|}$ and then output the first n bits of U(p, ε)
- We need to compute the probability that a computable process p is consistent with observation sequence x – this can now be done using Bayes' rule

Solomonoff Induction

- Solomonoff Induction provides a mathematically rigorous framework for induction
- Pros
  - Framework is conceptually clean and insightful
  - Mathematically rigorous guarantees can be shown on the accuracy of predictions
- Con
  - The induction process is itself uncomputable because Kolmogorov complexity is uncomputable
- However, by using computable approximations of Kolmogorov complexity, such as resource-bounded variants, Solomonoff induction has been implemented in practice by scientists such as Hutter and Schmidhuber

Takeaways

- Kolmogorov complexity is useful for providing foundations for learning in a very general sense
- This is done by using Solomonoff induction: Bayesian learning with a universal prior inspired by Occam's Razor
- Conceptually clean framework with mathematically rigorous guarantees, and variants of it have been shown to be useful in practice

Plan of Talk

- Motivation
- Intro to Kolmogorov Complexity
- Resource-Bounded Variants
- Learning
- *Cryptography*

Cryptography and Compression

- On the surface, it might not be obvious why crypto is related to the theory of compression
- However, *pseudo-random generators* are essential to encryption and other cryptographic tasks
  - A pseudo-random generator (PRG) maps short random "seeds" to longer "pseudo-random" strings that are *computational indistinguishable* from random strings
  - We can think of a PRG as a sort of de-compressor, and correspondingly the outputs of a PRG are in principle compressible
  - The security of a PRG relies on the compression not being doable efficiently

Cryptography and Compression

- Pseudo-randomness implies the hardness of compression, but can PRGs be based on the hardness of compression?
- A very recent line of work (by Liu-Pass, Ren-S, Ilango-Ren-S) aims to do precisely this, by basing PRGs (in fact, the equivalent notion of one-way functions) on the *average-case* hardness of time-bounded Kolmogorov complexity K^t

Average-Case Hardness

- Typically, complexity theory deals with worst-case hardness, i.e., a computational problem is considered hard if there is no efficient algorithm solving it correctly on *all* instances
  - But these hard instances may be rare or hard to find, so this is not always a satisfactory notion of hardness
- Average-case hardness studies hardness with respect to *distributions* on inputs
- A problem L is considered to be average-case hard over distribution D if no efficient algorithm solves L correctly with high probability on instances sampled from D

Average-Case Hardness Assumptions on K^t

- Recall that the K^t problem is the problem of computing the t-bounded Kolmogorov complexity of a string
- How do we model the average-case hardness of K^t?
  - Hardness over the *uniform* distribution is natural to consider
  - But more generally, we could consider hardness over *any samplable* distribution, i.e., a distribution sampled by a polynomial-time algorithm
- Remarkably, both of these notions of average-case hardness lead to *equivalences* with the existence of PRGs!

Pseudo-randomness is Equivalent to the Hardness of Compression

- Theorem [Liu-Pass]: PRGs exist if and only if K^t is hard over the uniform distribution
- Theorem [Ilango-Ren-S]: PRGs exist if and only if there is a samplable distribution D such that K^t is hard to approximate over D
- These results are the first ones to give an equivalence between PRGs and hardness for a *natural* problem, i.e., K^t
  - It is well-known that the hardness of problems such as Factoring or Learning with Errors implies the existence of PRGs, but these implications are not known to be equivalences

Takeaways

- There is an intuitive link between pseudo-randomness and compression – the outputs of PRGs are compressible in principle but this compression needs to be intractable in order for PRGs to be secure
- In recent work, this intuitive link has been turned into formal characterizations of pseudo-randomness in cryptography by average-case hardness of the K^t problem

Conclusion

- Motivated by the goal of creating a theory of compression for individual strings, we defined Kolmogorov complexity and its variants
- These notions turn out to be philosophically deep and have relevance to fundamental problems in many areas of computer science, including complexity theory, learning and cryptography
- They also lead to intriguing open problems

Open Problems

- Is K^t NP-hard to compute, for polynomially bounded t?
- Are there near-optimal compression schemes with poly-time computable compressors and de-compressors?