Logistics Review

Class Road Map - IT-I

- L19 (1/6): Overview, Communications, Gaussian Channel
- L20 (1/8): Gaussian Channel, band limitation, parallel channels, optimization and duality
- L21 (1/13): parallel channels, colored noise, feedback, matrix inequalities
- L22 (1/15): matrix inequalities, rate distortion.
  - (1/20): Monday holiday
- L23 (1/22): rate distortion for Bernoulli, Gaussian, and Multiple Gaussians with unequal noise
- L24 (1/27): main rate distortion theorem, geometry
- L25 (1/29): computing $R(D)$
- L26 (2/3): computing $R(D)$, alternating minimization
- L27 (2/5): Kolmogorov complexity, algorithmic randomness

- L28 (2/10):
- L29 (2/12):
  - (2/17): Monday, Holiday
- L30 (2/19):
- L31 (2/24):
- L32 (2/26):
- L33 (3/3):
- L34 (3/5):
- L35 (3/10):
- L36 (3/12):

Logistics Review

Cumulative Outstanding Reading

- Read Ch. 14 in our book (Cover & Thomas, “Information Theory”).
- Read Ch. 10 in our book (Cover & Thomas, “Information Theory”).
- Read Ch. 17 in our book (Cover & Thomas, “Information Theory”) on matrix inequalities.
- Read Ch. 9 in our book (Cover & Thomas, “Information Theory”)
- Read Ch. 5 in Boyd and Vandenberghe’s Convex Optimization book
- Read all readings assigned in EE514a, Fall 2013. (see later lectures on our previous web page (http://j.ee.washington.edu/~bilmes/classes/ee514a_fall_2013/)).

Homework

- No current outstanding HW.
Office hours on Mondays, 3:30-4:30.
As always, email me if you want to skype/google hangout rather than come to office hours, also at different times.

On Final Presentations

- Your task is to give a 10-15 minute presentation that summarizes 2-3 related and significant papers that come from IEEE Transactions on Information Theory (or a very related area).
- The papers must not be ones that we covered in class, although they can be related.
- You need to do the research to find the papers yourself (i.e., that is part of the assignment).
- The majority of the papers must have been published in the last 10 years (so no old or classic papers).
- Your grade will be based on how clear, understandable, and accurate your presentation is (and also milestones).
- This is a real challenge and will require significant work! Many of the papers are complex. To get a good grade, you will need to work very hard to present very complex ideas in an extremely simple yet still precise way.
- Again, don’t expect this to be easy, you might need to try a few topics until you find one that is suitable.
Final Presentation Milestones

All submissions done in PDF file format via our assignment dropbox (https://canvas.uw.edu/courses/880971/assignments)

- **Monday, Feb 17th, 11:45pm**: Candidate proposed papers submitted. Include short **at most** 1-page writeup: 1) why you chose these papers; 2) how they are related to each other; 3) why they are important to pure IT; and 4) how they are fundamental and/or deep, and 5) how will you summarize them in a simple and precise way.

- **Monday, Feb 24th 11:45pm**: Updated list of proposed papers decided, based on feedback. Updated writeup with more description.

- **Monday, March 3rd 11:45pm**: progress report (at most 1 page). Any background papers you needed to read to better understand your core set. Thoughts on coherent and simple unifying presentation.

- **Monday, March 10th, 11:45pm**: updated short (≤ 1 page) writeup on more details of how you will present the ideas in a simple fashion.

- **Final presentations**: Monday, March 17, 2014, 2:30–4:20pm, LOW 102. What to turn in: your slides and a short at most 4 page summary of the papers.

---

- The next set of five slides are from Lectures 2 and 3.
Why Entropy
Complexity
Kolmogorov
Algorithmic Randomness
Universal Prob.

Entropy, MI, CMI, 3 RVs, in a Venn Diagram

Given three random variables $X_1, X_2, X_3$ related by $p(x_1, x_2, x_3)$, the following Venn diagram characterizes the relationships.

- $I(X_1; X_2; X_3)$
- $H(X_2)$
- $H(X_2 | X_1)$
- $H(X_1)$
- $H(X_1 | X_2, X_3)$
- $I(X_1; X_3)$
- $H(X_3)$

Entropy

- Information - we use entropy to measure information.
- The communication theory model
- Surprise of an event $\{X = x\}$ is measured $\log \frac{1}{p(x)}$, and there are reasons for using $\log$.
- Entropy is lower bound on min number of guesses (on average) to guess the value of a random variable.
- Entropy is the minimum number of bits to compress a source.
- Entropy is the optimal coding “length”, of a random source.
- Entropy is the minimum description length (MDL) of a random source that can be achieved without probability of error.
Why log?

- We defined the information in event \( \{X = x\} \) as 
  \[ I(\{X = x\}) = I(x) = \log \frac{1}{p(x)}, \]
  but why log?
- Intuition says we want the information about an event to be
  inversely related to the probability, but there are many such
  relationships that might be useful.
- E.g., other possible functions include 
  \[ I(x) = (p(x))^{-1/n} \] for some 
  \( n > 0 \).
- Another example. \( I(x) = \) number of prime factors in \( [1/p(x)] \)
- But log, as well will see, has a number of attractions.

For a distribution on \( n \) symbols with probabilities 
\( p = (p_1, p_2, \ldots, p_n) \), let \( H(p) = H(p_1, p_2, \ldots, p_n) \) be the entropy of 
that distribution.

Consider any information measure, say \( S_I(p) \) on \( p \), and consider the 
following three natural and desirable properties.

- \( S_I(p) \) takes its largest value when \( p_i = 1/n \) for all \( i \).
- If we define the conditional information as
  \[ S_I(Y|X) \triangleq \sum_x p(x) S_I(p(y_1|x), p(y_2|x), \ldots, p(y_n|x)) \] (27.22)
  \[ = \sum_x p(x) S_I(Y|X = x), \] (27.23)
  then we wish to have additivity in the following way
  \[ S_I(X, Y) = S_I(X) + S_I(Y|X) \] (27.24)
- For a distribution on \( n + 1 \) symbols, then if the probability of one is
  zero, we wish for \( S_I(p_1, p_2, \ldots, p_n, 0) = S_I(p_1, p_2, \ldots, p_n) \)
Theorem 27.3.5 (Khinchin’s Theorem)

If $H(p_1, \ldots, p_n)$ satisfies the above 3 properties for all $n$ and for all $p$ such that $p_i \geq 0$, $\forall i$ and $\sum_i p_i = 1$ (i.e., all probability distributions), then

$$H(p_1, \ldots, p_n) = -\lambda \sum_i p_i \log p_i$$

(27.22)

for $\lambda$ a positive constant.

Thus, we get entropy for some logarithmic base.

Strings

Consider the strings:

- Consider 0101011. While this is an infinitely long string, and while there is no probability distribution governing it (it is one string after all), it is somehow simple to describe. For example, we can just say “Print 01 forever” which is a 16 byte description of an infinitely long string. Alternatively, we can say “0101” which is a, perhaps, 5 byte description of the string.

- Consider

$$11.001001000111110111010101000100010110100011000100011010011\ldots$$

Actually, this is a binary representation of $\pi$. It has no known periodicity or repeating pattern. Is it easy or hard to describe? There is a simple algorithm to generate this string to any length, and the algorithm is fixed length. We could think of the algorithm itself as a code for this string. Whenever we want to communicate the string, we instead just send the algorithm to generate it. The computer then, in some sense, becomes the “codebook.”
Strings

- Consider $b_1 b_2 b_3 \ldots b_n$ where this was generated via stochastic process $p(b_i = 1) = 1 - p(b_i = 0) = 0.5$ and $b_i$ are i.i.d. Is there a short way to describe this string? No, this string is incompressible. The only way to send the string (or compress it) is to send (store) it directly.

String Complexity

Consider the complexity of a string to be the length of an algorithm needed to generate that string. I.e., for the examples above, we’d have:

- Repeat: Print 01; which is very short
- Print $\pi$; which is longer, but still relatively simple.
- Print $b_1 b_2 b_3 \ldots$; The program is long since the string needs to be embedded as an immediate operand to the program.
On Program Length

- We don’t care about what language we use since, if we have a universal computing device, all computers can simulate each other.
- A Turing machine (a mathematical model for computation) consists of an input stream (tape, read sequentially), a working tape of infinite length (memory), and an output stream (tape also).

Properties: 1) An infinite “tape” (or memory); 2) read/write head (can read and write to memory); 3) machine has a finite number of states (FSM); 4) a transition function \( \delta \) which does one of:

\[
\delta(\text{current state}, \text{read input}) =
\begin{cases} 
\text{read/write on work tape} \\
\text{write in output} \\
\text{move to a new state}
\end{cases}
\]  

Turing Machine

- All existing standard computing models can be reduced to Turning machine.
- Church Thesis: A Turing machine can compute any function commutable by a “reasonable” computing device with only polynomial cost over any other.
- Reasonable does not include quantum models of computation.
- Thus, we make the following assumption:

**Lemma 27.4.1**

All computing devices (at least that we deal with) are equivalent to a universal computer. \( \mathcal{U} \) will designate this universal computer.

- We’ll use \( p \) in this context to designate a program and the length of this program, as measured by \( \mathcal{U} \) will be \( \ell(p) \). \( \mathcal{U}(p) \) is \( \mathcal{U} \) run on program \( p \) which will produce some output (say \( x \)).
Kolmogorov Complexity

- We are now ready to define another notion of complexity of an object that is entirely non-probabilistic.

**Definition 27.5.1 (Kolmogorov Complexity)**

The Kolmogorov Complexity of a string $x$ measured according to $\mathcal{U}$ is defined as:

$$K_{\mathcal{U}}(x) = \min_{p : \mathcal{U}(p) = x} \ell(p)$$  \hspace{1cm} (27.2)

So $K_{\mathcal{U}}(x)$ is the shortest algorithmic description length of the string $x$ over all algorithmic descriptions that generate $x$ by computer $\mathcal{U}$.

Non-Halting

- The definition requires that $p$ halts, meaning that it eventually will terminate (even if it takes a long time).
- Not all programs will halt and there are reasons (as we will see) for excluding these non-halting problems.
- A natural number (non-negative integer) $k$ can be written as the sum of $n$ squares of it is possible to write $k = \sum_{i=1}^{n} b_i^2$ for non-negative integers $\{b_i\}_i$.
- Consider the problem of computing the smallest natural number that is not the sum of $n$ squares ($0^2 = 0$ included).
- So if $p$ is a program that does this iteratively (by trying out each number), then $p(0) = 1$, $p(1) = 2$, $p(2) = 3$, and $p(3) = 7$ (since $8 = 0^2 + 2^2 + 2^2$). Thus, for these arguments, $p$ will halt.
- What is $p(4)$? If we present $p$ to a turning machine then $\mathcal{U}(p, 4)$ does not halt, since Lagrange showed that every number is the sum of four square numbers.
- Thus $p$, when given 4 as input, does not halt.
**Example**

- Consider sequences/strings of large length, e.g., $\ell(x) \approx 10^{15}$ (in this case, $\ell$ measures the length of string $x$).
- One possible case 1: Expand $e$ to $10^{15}$ bits. Is this complex (via $K_U$)? No, since we can write a short program to do this.
- Case 2: are most strings of length $10^{15}$ complex or simple (by $K_U$)?
  - A: Complex. Why? How many simple strings can there be of a given length? No more than the number of short programs, and there can’t be too many of those.
- Intuition: AEP. Recall $P_n \leq (n + 1)^{|X|}$, $Q^n(x) = 2^{-n(D(P_x||Q)+H(P_x))}$, $|T(P)| = 2^nH(P)$, $Q^n(T(P)) \approx 2^{-nD(P||Q)}$.
- So for $x$ a string with type $P_x$, we have
  $$|T(P_x)| = 2^nH(P_x)$$
  meaning that there are exponentially more sequences of higher entropy than there are of lower entropy. But this is just entropy ...

**Conditional Kolmogorov Complexity**

**Definition 27.5.2**

The conditional Kolmogorov complexity is defined for a string $x$ given $y$ as input that is not charged in the length. I.e.,

$$K_U(x|y) = \min_{p : \mathcal{U}(p, y) = x} \ell(p) \quad (27.4)$$

- So here, the program can use $y$ as input, and the length of $y$ is not charged against the length of the program $p$.
- $K_U(x|x)$ = constant, since if we have $x$ we can just have a program that says print my input.
- $K_U(x|y)$ when $y$ is i.i.d. and $x$ is also i.i.d. will be such that $K_U(x|y) = K_U(x)$.
Conditional Kolmogorov Complexity

Definition 27.5.3

The **length conditional** Kolmogorov complexity is defined for a string \( x \) as follows:

\[
K_U(x|\ell(x)) = \min_{p : U(p, \ell(x)) = x \text{ and } p \text{ halts}} \ell(p)
\]  

(27.5)

- So program has more info available to it, but only the length of what it must generate.
- Does this increase or decrease Kolmogorov Complexity? More soon...

Universality

Theorem 27.5.4

Kolmogorov complexity is universal. I.e., given universal computer \( U \) and another computer, say \( A \), then

\[
K_U(x) \leq K_A(x) + c_A
\]  

(27.6)

for all \( x \in \{0, 1\}^* \), and where \( c_A \) is a constant and does not depend on \( x \).

- Consider program \( s_A \) to simulate \( A \) by \( U \) (so that \( U(s_A, p_A) = A(p_A) \)).
- Given \( p_A \) s.t. \( A(p_A) = x \), then \( U(s_Ap_A) = x \).
- But \( \ell(s_Ap_A) = \ell(s_A) + \ell(p_A) = c_A + \ell(p_A) \).
- Thus, since \( K_U(x) \leq \ell(s_Ap_A) \) for all \( p_A : A(p_A) = x \), we have

\[
K_U(x) = \min_{p : U(p) = x} \ell(p) \leq \min_{p : A(p) = x} (\ell(p) + c_A) = K_A(x) + c_A
\]  

(27.7)

- Hence, we ignore which computer it is (unless otherwise stated).
**Theorem 27.5.5**

\[ K(x|\ell(x)) \leq \ell(x) + c \]  

(27.8)

where \( c \) is constant.

**Proof.**

- The computer knows \( \ell(x) \), so consider the program:
  
  \[
  \text{print the sequence } x_1: \ell(x)
  \]

  (27.9)

- This program has length \( \ell(x) + c \).
- No additional bits are needed to store the value \( \ell(x) \) itself, since that is an input to the program.

- Often not a good upper bound, as this is really the worst case.

**Comparing w. Conditioning**

**Theorem 27.5.6**

\[ K(x) \leq K(x|\ell(x)) + 2 \log \ell(x) + c \]  

(27.10)

where \( c \) is constant.

**Proof.**

- In worst case, for \( K(x) \), need a way of representing both the string \( x \) and its length \( \ell(x) \) in program.
- Can’t just use \( \log n \) bits to store a number \( n \), still need to know how many bits we use to store a number, a meta regress problem.
- Instead, express representation of \( \ell(x) \) using \( 2\lceil \log n \rceil \) bits, duplicating bits, and then use 01 as code for an “end” symbol.
- I.e., 5 = 101 gets represented as 11001101 where the final 01 is the end symbol.
Comparing w. Conditioning

Theorem 27.5.7

\[ K(x) \leq K(x|\ell(x)) + \log^* \ell(x) + c \quad (27.11) \]

where \( \log^* n = \log n + \log \log n + \log \log \log n + \ldots \) and this continues until just before the term goes negative.

- This is just another way to encode a number.
- We use \( \log n \) bits to store a number \( n \), but we need to know many bits are being used to store the number \( \log n \) itself,
- Then we use \( \log \log n \) bits store a number \( \log n \), but we need to know how many bits are being used to store \( \log \log n \),
- etc.

Complexity and counting

- So complexity is measured based on the shortest program that can generate it.
- There are many more (exponentially more) long programs than there are short programs. Why? Since short programs, themselves, take up fewer bits.
- Thus, there can't be more strings that have short programs then there are short programs to begin with,
- Thus, unfortunately, most strings are complex, which we formalize next.
**Theorem 27.5.8**

The number of strings $x$ with $K(x) < k$ is

$$|\{x \in \{0, 1\}^* : K(x) < k\}| < 2^k \quad (27.12)$$

**Proof.**

- How many programs are there with length $< k$? A: $= 2^k - 1 < 2^k$.
- I.e., there is 1 program with length 0; there are 2 program (0 and 1) with length 1; there are 4 programs (00, 01, 10, 11) with length 2; 8 program with length 3, etc.
- In general, $2^k$ program with length $k$
- $\sum_{\ell=1}^{k-1} 2^k = 2^k - 1 < 2^k$
- Each program (at best) outputs a unique string $x$.
- Thus, there can be at most $2^k$ programs with complexity $< k$.

Note, if we look at all $2^k$ programs of length $< k$, and don’t find one such that $U(p) = x$, then $K(x) \geq k$.

Finding the shortest program in this way has computational cost exponential in $K(x)$, but we’ll discuss this more soon.
Examples

- $K(00\ldots0|n) = c$ for all $n$.
- $K(\pi_1\pi_2\ldots\pi_n|n) = c$ for all $n$
  
  bit expansion of $\pi$
- $K(0101|n) = c$, i.e., “Print 01 $n/2$ times”

Examples: Weather prediction (does it rain or not).

- Given binary string $x_{1:n}$ with $x_i = 1$ iff it rains, otherwise $x_i = 0$.
- We model this as a Markov chain, and learn (from $x$) a conditional histogram $p(x_i|x_{i-1})$ which requires 4 table entries.
- Consider histogram of counts of pairs, numerator of histogram is at most $n$, so $\log n$ bits needed. Thus, $O(\log n)$ bits for $p(x_i|x_{i-1})$.
- We need $\log 1/p(x_i|x_{i-1})$ bits to describe symbol $x_i$ in context of $x_{i-1}$ (e.g., arithmetic coding).
- Thus, the Kolmogorov Complexity is:
  
  $$K(x_{1:n}|n) = Hn + O(\log n) + c$$  \hspace{1cm} (27.13)

  $Hn$ bits are needed for the string ($H$ is the per symbol entropy), $O(\log n)$ bits needed for the table, and $c$ is a constant.
- Seattle case:
  
  $$K(x_{1:n}|n, Seattle) = c$$  \hspace{1cm} (27.14)

  i.e., program is “while(1) { print rain; }”
Example: Fractal Image

- For all $c \in \mathbb{C}$ image where $c$ is a complex number indexing a pixel element, find $n: z_{n+1} = z_n^2 + c$, and $z_{n+1} > \tau$. Resulting $n$ for $c$ then indexes the color of pixel $c$.
- Resulting images looks very complex (r.h.s. is other form of fractal).
- but, $K(\text{fractal image}|\text{num. pixels}) = \text{const.}$
- Natural images? Is there a simple algorithm to generate them?

More Examples

- $K(n) \leq \log^* n + c$ for integer $n$ (again, $\log n$ bits to store $n$, $\log \log n$ bits to store $\log n$, etc.).
**More Examples**

- Any $x_1:n$ s.t. $\sum_i x_i = k$, number of binary strings with $k$ 1's. The question is: how can we encode this?
  - Consider strings of length $n$ with $k$ 1’s, sorted in lexicographic order.
  - Given index $i$, can write a program with constant length to print out the $i$'th one, i.e., “print out the $i$'th string if the $i$'th one = $x_1:n$.”
  - So given this program, we need only represent both $k$ and $i$.
  - For $k$, need $2 \log k$ bits (or could use $\log^* k$ bits, or even $\log n$ bits).
  - For $i \in [1, \ldots, \binom{n}{k}]$, need $\log \binom{n}{k}$ bits. Why no 2? Since we have both $n$ (as input) and $k$, so we know number of bits needed for $i$.
  - So the length of the program is:
    \[
    \ell(p) = c + 2 \log k + \log \binom{n}{k} \quad (27.15)
    \leq c + 2 \log k + nH(k/n) \quad (27.16)
    \]
  - Since $\binom{n}{k} \leq 2^{nH(k/n)}$
  - Thus,
    \[
    K(x_1:n) \leq 2 \log k + nH(k/n) + c \quad (27.17)
    \]
  - Note, if $k = n/2$, this is as long as the string itself.

---

**$K$ vs. $H$**

- As we saw, in a few examples, the expression for $K(x)$ involved the entropy. How do they compare more generally?
  - First point: $H$ requires a distribution while $K$ does not (but then neither does Lempel-Ziv compression which, we can show, while algorithmic will converge to the entropy).
  - First, an important theorem.

**Theorem 27.5.9**

∀$U$ (meaning under any “reasonable” computing device),

\[
0 \leq \Omega \triangleq \sum_{p: U(p) \text{ halts}} 2^{-\ell(p)} \leq 1 \quad (27.18)
\]

**Proof.**

If $p$ halts, then it can’t be a prefix of any other program. Therefore, halting programs satisfy the prefix property, and we immediately have countably infinite Kraft inequality (from Lecture 9).
Theorem 27.5.10

Let \( \{X_i\}_i \) be i.i.d. drawn \( \sim p(x) \) for \( x \in \mathcal{X} \). Also, \( p(x_{1:n}) = \prod_i p(x_i) \).
Then \( \exists c \) s.t.

\[
H(X) \leq \frac{1}{n} \sum_{x_{1:n}} p(x_{1:n}) K(x_{1:n} | n) \leq H(X) + \frac{\mathcal{X}}{n} + \frac{c}{n} \tag{27.19}
\]

Also,

\[
E \frac{1}{n} K(X_{1:n} | n) \to H(X) \tag{27.20}
\]

So, \( K(x_{1:n} | n) \) is, on average, and when \( n \) gets big, no better than the codeword lengths suggested by \( \log 1 / p(x_{1:n}) \).

Proof of Theorem 27.5.10.

- **Lower bound**: Consider error-free coding \( x_{1:n} \) with codewords of length \( \ell(x_{1:n}) \).
- We know in such case \( \sum_{x_{1:n}} p(x_{1:n}) \ell(x_{1:n}) \geq H(X_{1:n}) \)
- Let’s code \( x_{1:n} \) with \( \arg \min_{U} p_{U}(x_{1:n}) = x \ell(x) = p^*(x) \), so that \( \ell(p^*(x)) = K(x | n) \) which as we saw above satisfies Kraft inequality.
- Therefore,

\[
\sum_{x_{1:n}} p(x_{1:n}) K(x_{1:n} | n) \geq H(X_{1:n}) = nH(X) \tag{27.21}
\]

- **Upper bound**: We encode the “type” of the string, and then encode which (in lexicographic order) of the strings it is with that type.
Proof of Theorem 27.5.10.

- Consider the type of $x_{1:n}$:
  \[ P_{x_{1:n}} = \left( \frac{n(a_1)}{n}, \frac{n(a_2)}{n}, \ldots, \frac{n(a_{|X|})}{n} \right) \quad (27.22) \]

- So, for the type, we need $|X| \log n$ bits to describe the type.
- Then, we enumerate which element is within the type class, and use the result $|T(P_x)| = 2^n H(P_x)$
- Thus, need $nH(P_x)$ bits for encoding which entry in type class.
- Thus, we have
  \[
  K(x_{1:n}|n) \leq nH(P_x) + |X| \log n + c
  \]
  \[
  \Rightarrow \quad EK(X_{1:n}|n) \leq nE[H(P_X)] + |X| \log n + c
  \quad (27.24)
  \]

Proof of Theorem 27.5.10.

- Note that $n_i(x_{1:n}) = n_i = \sum_{j=1}^{n} 1_{\{x_j = i\}}$ so that $En_i(X_{1:n}) = p_i n$.
- Also, recall $H(p)$ is concave in $p$, so $E[H(P)] \leq H(E[P])$.
- The last part of the proof follows from the following derivations:
  \[
  EH(P_X) = -\sum_{x_{1:n}} p(x_{1:n}) \left( \sum_i \frac{n_i}{n} \log \frac{n_i}{n} \right)
  \leq -\sum_i \frac{\sum_{x_{1:n}} p(x_{1:n})n_i}{n} \log \frac{\sum_{x_{1:n}} p(x_{1:n)}n_i}{n}
  \leq -\sum_i \frac{p_i n}{n} \log \frac{p_i n}{n}
  \]
  \[
  = H(X)
  \quad (27.28)
  \]
Algorithmic Randomness

- What is randomness? Normally, we define randomness via the presumption of the existence of certain probabilities. But the existence of probabilities, or what they mean, is not universally agreed upon (e.g., frequentist vs. Bayesian interpretation).
- We can dispense with probabilities and define algorithmic randomness of $x$ to be an inability to find a short program to print $x$.
- We can relate algorithmic randomness to our “normal” notion of randomness such as entropy, proportion of outcomes (typical sets), etc.
- How many simple sequences are there? Not too many. How many complicated sequences are there? Lots! We can look at this randomness as, in some sense, a surrogate for counting and ratios.

Bernoulli model of source

Theorem 27.6.1

Let $X_1, \ldots, X_n$ be $\sim$ Bernoulli(1/2). Then

$$\Pr(K(X_{1:n}|n) < n - k) < 2^{-k}\quad (27.29)$$

Proof.

$$\Pr\left(K(X_{1:n}|n) < n - k\right) = \sum_{x_{1:n}: K(x_{1:n}|n) < n - k} \Pr(x_{1:n})\quad (27.30)$$

$$= \sum_{x_{1:n}: K(x_{1:n}|n) < n - k} 2^{-n}\quad (27.31)$$

$$= |\{x_{1:n} : K(x_{1:n}|n) < n - k\}| 2^{-n} \quad (27.32)$$

$$< 2^{n-k} 2^{-n} = 2^{-k}\quad (27.33)$$
### Bernoulli model of source and Algorithmically Random

- So, as we increase $k$ (i.e., compress $x_{1:n}$ by $k$ bits, from $n$ bits to $n-k$ bits), the probability of being able to do so decreases exponentially fast!
- Multiply the probabilities by $n$, and we get that the number of sequences of length $n$ that can be compressed by $k$ bits decreases exponentially fast in $k$!

**Definition 27.6.2**

$x_{1:n}$ is algorithmically random if

$$K(x_{1:n}) \geq n - c_n \text{ with } c_n \to 0 \text{ as } n \to \infty$$

(27.34)

- So this is really a definition of a sequence of strings $\{x_{1:n}\}_n$ (or alternatively, an infinite length string that is considered at each point truncated to length $n$), such that, when $n$ gets big, the string becomes ever less compressible.

### Compressibility

**Definition 27.6.3**

A binary string $x_1, x_2, \ldots$, infinite in length, is algorithmically incompressible if:

$$\lim_{n \to \infty} \frac{K(x_1, x_2, \ldots, x_n | n)}{n} = 1$$

(27.35)

- So this is like $K(x_{1:n} | n) = n + c$.
- We know that if a random string is incompressible in an entropic sense, there are likely to be about half 0’s and half 1’s (i.e., the probability is close to 1/2). Is there an analogous notion for Kolmogorov complexity?
Compressibility

Theorem 27.6.4 (strong law)

If \( x_{1:\infty} \) is incompressible, then

\[
\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} x_i = \frac{1}{2}
\]

(27.36)

Like a series of i.i.d. variables where \( p(X_i = 1) = \frac{1}{2} \), but note that we’ve said nothing about probabilities, rather we’ve only mentioned its algorithmic complexity.

Proof.

- Define \( \theta_n = \frac{1}{n} \sum_{i=1}^{n} x_i \), then from Eq (27.23),

\[
n - c_n \leq K(x_{1:n} | n) \leq nH(\theta_n) + 2 \log n + c'
\]

(27.37)

with \( c_n/n \to 0 \). Therefore,

\[
H(\theta_n) \geq 1 - \frac{2 \log n + c_n + c'}{n}
\]

(27.38)

... proof continued.

- But \( H(\theta_n) \leq 1 \) and

\[
\frac{2 \log n + c_n + c'}{n} \to 0
\]

(27.39)

- Thus, eventually \( H(\theta_n) \to 1 \) meaning \( \theta_n \to \frac{1}{2} \).

So, note we’ve taken an non-random sequence and used entropy (which is defined on a probability distribution). Is this circular?

- No. We are using the distribution only as a combinatorial “counting” quantity, it is normalized so that we can still take advantage of the capabilities we know (and love 😊) about entropy.

- Akin to the method of types, and is also akin to the various proofs of purely combinatorial and/or matrix bounds using entropic quantities for assistance (known as the probabilistic method).
Also, truly algorithmically incompressible sequences all look random, and satisfy all statistical tests for randomness.

Why all? If not all, we can use the test to reduce the description length. I.e., suppose for certain \( i \), \( x_i \) and \( x_{i+k} \) are more likely to be 11 than 00, 01, 10, then we can group sequences into two equal sized groups with different probabilities ⇒ compressible.

Next, for random strings, we can get a direct relationship between Kolmogorov complexity and entropy, as we do next:

Theorem 27.6.5

Let \( X_1, X_2, \ldots, X_n \) be i.i.d. Bernoulli(\( \theta \)). Then

\[
\frac{1}{n} K(X_{1:n}|n) \to H(\theta) \text{ in probability} \tag{27.40}
\]

meaning

\[
\Pr(|\frac{1}{n} K - H| > \epsilon) \to 0 \quad \forall \epsilon > 0
\]

We’ll show:

- A: \( \Pr(K > n(H + \epsilon)) \to 0 \) as \( n \to \infty \) for any \( \epsilon > 0 \), and
- B: \( \Pr(K < n(H - c)) \to 0 \) as \( n \to \infty \) for any \( c > 0 \) (\( c \) is used as an \( \epsilon \) like quantity but we use symbol \( c \) since we’ll also use \( \epsilon \) in the same context below).
proof of Theorem 27.6.5.

- First (A), define $\bar{X}_n \triangleq \frac{1}{n} \sum_{i=1}^{n} X_i$, the probability of 1's in $X_{1:n}$.
- From previous lecture, we have that

$$K(x_{1:n} | n) \leq nH(\bar{x}_{1:n}) + 2 \log n + c \quad (27.41)$$

or

$$\frac{K}{n} - \frac{2 \log n + c}{n} \leq H(\bar{x}) \quad (27.42)$$

We then get the following:

$$\Pr \left( |\bar{X} - \theta| > \epsilon \right) \to 0 \text{ as } n \to \infty \quad \forall \epsilon > 0 \quad (27.43)$$

$$\Rightarrow \Pr \left( |H(\bar{X}) - H(\theta)| > \epsilon \right) \to 0 \quad (27.44)$$

$$\Rightarrow \Pr \left( -\epsilon + H(\theta) < H(\bar{X}) < \epsilon + H(\theta) \right) \to 1 \quad (27.45)$$

$$\Rightarrow \Pr \left( \frac{K}{n} - \frac{2 \log n + c}{n} < \epsilon + H(\theta) \right) \to 1 \quad (27.46)$$

$$\Rightarrow \Pr \left( \frac{K}{n} - H(\theta) \geq \epsilon \right) \to 0 \quad (27.47)$$

$$\Rightarrow \Pr \left( K > n(H + \epsilon) \right) \to 0 \quad (27.48)$$

Which is the first part.
Compressibility

proof of Theorem 27.6.5 continued.

For \( B \) (i.e., \( \Pr(K < n(H - c)) \to 0 \)). We have, for all \( \epsilon, c > 0 \):

\[
\Pr(K(X_{1:n}|n) < n(H(\theta) - c)) = \sum_{x_{1:n} : K(x_{1:n}|n) < n(H(\theta) - c)} \Pr(x_{1:n})
\]

\[
= \sum_{x_{1:n} \notin A_{\epsilon}^{(n)}, K(x_{1:n}|n) < n(H(\theta) - c)} \Pr(x_{1:n})
+ \sum_{x_{1:n} \in A_{\epsilon}^{(n)}, K(x_{1:n}|n) < n(H(\theta) - c)} \Pr(x_{1:n})
\]  \( \text{(27.49)} \)

\[
\leq \Pr(A_{\epsilon}^{(n)c}) + \sum_{x_{1:n} \in A_{\epsilon}^{(n)}, K(x_{1:n}|n) < n(H(\theta) - c)} \Pr(x_{1:n})
\]  \( \text{(27.50)} \)

\[
\leq \epsilon + 2^{-n(H(\theta) - \epsilon)}
\]  \( \text{(27.51)} \)

which can be made arbitrarily small for appropriate \( \epsilon, c, \) and \( n \).

Note, in the above case \( B \), we first choose a \( c > 0 \), and then choose an \( 0 < \epsilon < c \) so that for large enough \( n \) the above will \( \to 0 \).
Suppose $H = 1$, so that the number of typical sequences of length $n$ is $2^{nH} = 2^n$, so all sequences are equally likely.
Thus, $p(0, 0, \ldots, 0) = \frac{1}{2^n}$ where we have $L$ 0's.
Also, $p(1011011011101\ldots01101) = \frac{1}{2^n}$.
Both the all zeros, and the seemingly “random” sequence are in the typical set and are thus considered equally likely?
Which one is more surprising to you intuitively? Which one is more simple? Which one is easier to describe?
Is Shannon’s theory lacking? It only gives complexity to an ensemble of messages (or a diversity of messages, not to the message contents). Shannon’s approach does not ask: How complex is the message $x_{1:n}$ itself?
Shannon says that the “complexity” of the single message is the coding length $\log \frac{1}{p(x_{1:n})}$, irregardless of $x_{1:n}$.
But how can we get to coding length without probabilities? We appeal to philosophy and Occam’s razor.

An Appeal to Philosophy

Occam’s razor: “If there are multiple explanations for a phenomenon, then all other things being equal, we should select the simplest one”
“simplicity of an object” is the same as “the object has the shortest effective description”
Occam’s razor says that the short explanations, or causes, of objects are the more probable ones.
This is really a prior probability on nature itself.
We arrange in our thought all possible events in various classes; and we regard as extraordinary those classes which include a very small number. In the game of heads and tails, if heads comes up a hundred times in a row then this appears to us extraordinary, because the almost infinite number of combinations that can arise in a hundred throws are divided:

1. in regular sequences, or those in which we observe a rule that is easy to grasp, and
2. in irregular sequences, that are incomparably more numerous.

Philosophical Essay on Probabilities, Laplace, 1819

Of course, Laplace wrote this 150 years before Kolmogorov complexity was formalized. How do we define “regular”? Does this exhibit a real-world bias, a bias towards the type of regular events we see in our universe?

The regular combinations occur more rarely only because they are less numerous. If we seek a cause wherever we perceive symmetry, it is not that we regard the symmetrical event as less possible than the others, but since this event ought to be the effect of

1. a regular cause [or short program] or
2. that of chance,

the first of these suppositions is more probable than the second. On a table we see letters arranged in this order “Constantinople” and we judge that this arrangement is not the result of chance, not because it is less possible than others, for if this word were not employed in any language we would not suspect it came from any particular cause, but this word being in use among us, it is comparably more probable that some person [program] has thus arranged the aforesaid letters than that this arrangement is due to chance.

Philosophical Essay on Probabilities, Laplace, 1819
Regular Sequences

- 000011110000111100001111 This is just repeat 4 0's and 4 1's for ever. Is this likely to happen by pure chance?
- 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 ... This is just the primes. Is this likely to happen by chance?
- 0 1 2 3 1 1 3 3 5 9 1 2 1 2 5 8 7 ... Let $p_i$ be the $i$'th prime. Then this sequence is $n_i = p_i \mod i$. Is this likely to happen by chance?
- Pulsars: From Wikipedia: in 1968: “An entirely novel kind of star came to light on Aug. 6 last year and was referred to, by astronomers, as LGM (Little Green Men). Now it is thought to be a novel type between a white dwarf and a neutron [sic]. The name Pulsar is likely to be given to it. Dr. A. Hewish told me yesterday:” We are more likely to believe this latter simpler explanation, although noone knows for certain.
- So it seems perhaps intuitive that simple explanations are more probable for a given phenomena, if they exist, than the alternative that they happen by chance.

Prefix-free programs

- Can we derive a probability distribution for this?
- We can quantify this by defining a probability on random programs that are prefix-free (halting). That is, for program $p$, define:

$$Pr(p) = 2^{-\ell(p)}$$

so that $\sum_p Pr(p) \leq 1$ by Kraft inequality.
- This is an assumed fundamental property of nature.
- Short programs are much more probable than long ones.
- Most sequences $x_{1:n}$ of length $n$ are complex (since there are just not enough short programs to go around to cover all the long sequences, of which there are many).
- When short programs produce long strings, they produce simple ones (by definition).
Universal Probability

- We can use this to produce a probability distribution on strings that is universal, where simple strings are more probable than complex ones of the same length.

**Definition 27.7.1**

The universal probability of a string $x$ is given by:

$$P_U(x) = \sum_{p: U(p) = x} 2^{-\ell(p)} = \Pr(U(\mathcal{P}) = x)$$

(27.54)

where on the r.h.s. $\mathcal{P}$ is a random program variable under $\Pr(\mathcal{P} = p)$.

- This is also equal to the probability that random fair coin flips will create a program to print out $x$.

The Meaning Of Life?

- $x$ could be natural images, natural scenes, or even sentience.
- And $p$ the program is what causes $x$ to occur.
- If nature acts as a grand computer in response to some random input, and if short programs are more probable than long ones, then this could cause $x$ to have structure even if $x$ is very big itself.
- Fractals might be an example of this but it might be only the most mundane of examples.
Computing Kolmogorov Complexity

- The Kolmogorov Complexity of a string $x$ is:
  \[ K_U(x) = \min_{\text{p} : U(p) = x} \ell(p) \quad (27.55) \]
  and $p$ halts

- We have found bounds on $K_U(x)$, but how to compute it?
- We could try all short programs and see which one of them (the first one) generates $x$.
- Some short programs might take a long time to halt.
- How do you know if a program will halt or not, since if it doesn’t halt we needn’t bother with this program.
- On the other hand, if it hasn’t halted yet, how do you know if we just haven’t waited long enough?
- We thus need a way to tell if $p$ will halt.

Halting Theorem

**Theorem 27.7.2**

There is no program $h(p, y)$ that takes a program $p$ with input $y$, where $h(p, y) = 1$ if $p$ halts on input $y$, and $h(p, y) = 0$ otherwise.

**Proof.**

- Suppose, to the contrary, that $\exists$ such an $h$,
- and given $h$ we can easily construct program $\Psi(x)$ as follows:
  \[ \Psi(x) = \begin{cases} 
  1 & \text{if } h(x, x) = 0 \quad \text{i.e., } x \text{ does not halt on } x. \\
  \text{spin forever} & \text{if } h(x, x) = 1 \quad \text{i.e., } x \text{ halts on } x.
  \end{cases} \quad (27.56) \]

$\Psi(x)$ asks: does $x$ not halt on itself as input? If it does halt, we spin forever. I.e., $\Psi(x)$ spins forever if $x$ halts on itself, and halts if $x$ spins forever on itself.
Halting Theorem

proof of Theorem 27.7.2.

- Now, consider constructing:

\[
\Psi(\Psi) = \begin{cases} 
1 & \text{if } h(\Psi, \Psi) = 0 \\
\text{spin} & \text{if } h(\Psi, \Psi) = 1 
\end{cases}
\]  

(27.57)

- But if \( h(\Psi, \Psi) = 0 \), then this implies \( \Psi(\Psi) \) spins forever by the definition of \( h \), contradicting the definition in first line of \( \Psi \) above.

- And if \( h(\Psi, \Psi) = 1 \), then \( \Psi(\Psi) \) halts, by definition of \( h \), contradicting the 2nd line of \( \Psi \) above.

- I.e., \( \Psi(\Psi) \) spins forever if \( \Psi \) halts on itself, and halts if \( \Psi \) spins forever on itself, i.e., a contradiction.

- Thus, we conclude that our assumption (\( h \) exists) must be false.

This is an instance of Gödel’s incompleteness theorem, very loosely saying that any theory of arithmetic that is sufficiently rich, some arithmetical truths are such that the theory cannot prove them. Important in mathematics, logic, and philosophy. See http://en.wikipedia.org/wiki/Godel’s_incompleteness_theorems

- Simple example: contradictions:
  
  - **Statement A**: “Statement B is false.”
  - **Statement B**: “Statement A is true.”

- Other example, the halting problem mentioned above.
Computing Kolmogorov Complexity

- Thus, $K_U(x)$ is not computable, since we cannot solve the halting problem.
- Suppose, just for fun, that we could compute $\Omega$.

$$0 \leq \Omega \triangleq \sum_{p \in U(p) \text{ halts}} 2^{-\ell(p)} \leq 1 \quad (27.58)$$

again which follows from the Kraft inequality.
- Key point: $0 < \Omega \leq 1$ actually exists (it is merely a number between 0 and 1) but we can’t compute it due to the halting theorem (this is actually an instance of Gödel’s theorem).
- But what if we could compute $\Omega$? What would this number tell us?
- Note, $\Omega$ is the halting probability, the probability that a program (drawn from the universal probability of programs) will halt.

Proving $n$-bit theorems

- What could we do, given $\Omega$?
- Given $\Omega$, we could prove any $n$-bit theorem?
- How? To do this for $n$-bit theorems, compute $\Omega_n = \lfloor \Omega \rfloor_n$ which is $\Omega$ truncated to $n$ bits.
- Next, run all programs simultaneously, $n$ cycles at a time.
- That is, given enumerable (countable number of) functions $p_1, p_2, p_3, \ldots$ run:

```plaintext
1 for i = 1, ... do
2   for j = 1, ..., i do
3     run i more cycles of $p_j$;
```
Proving \(n\)-bit theorems

- Once the sum of programs that have halted are such that
  \[
  \sum_{p \text{ halted so far}} 2^{-\ell(p)} \geq \Omega_n \tag{27.59}
  \]
  then no other program of length \(< n\) could halt, and we have solved the halting problem for any program of arbitrary length (up to length \(n\))
- How to use the halting problem to solve any hard conjecture?
- Here’s how to prove that \(P = NP\) or \(P \neq NP\).
- Let \(p\) be a program to search all programs that solve some NP-complete problem and halt only if the solution is polynomial time.
- Then if \(\text{halt}(p)\) is true, then \(P = NP\).

Compressibility of \(\Omega\)

Even if we could compute \(\Omega\), it is incompressible.

**Theorem 27.7.3**

\(\Omega\) cannot be compressed by more than a constant. I.e., there exists a constant \(c\) s.t.

\[
K(\Omega_n) = K(\omega_1 \omega_2 \ldots \omega_n) \geq n - c \quad \forall n \tag{27.60}
\]

**Proof.**

- Given \(\Omega_n\), we can solve the halting problem for any program of length \(\leq n\) bits.
- We can compute \(\Omega_n\) with a length \(K(\omega_1 \omega_2 \ldots \omega_n) = K(\Omega_n)\)-bit program.
- Generate list of all programs length \(\leq n\) that halt, and their output:
  \[
  (p_{h_1}, x_{h_1}), (p_{h_2}, x_{h_2}), \ldots, (p_{h_m}, x_{h_m}) \tag{27.61}
  \]
Compressibility of $\Omega$

... proof continued.

- Find the shortest string $x_0$ not in this list of output strings and output $x_0$.
- The above “program” (i.e., the enumeration of programs and output of $x_0$) has length $K(\Omega_n) + c$ (i.e., $K(\Omega_n)$ to compute $\Omega_n$ and $c$ to do the enumeration.
- We can’t have that $K(x_0) \leq n$ since we have enumerated all programs of length $\leq n$ that halt, and $x_0$ was not among them, so $K(x_0) > n$.
- The above program must also have length $\geq K(x_0)$ since it is a program to compute $x_0$. Thus,

$$K(\Omega_n) + c \geq K(x_0) > n$$

(27.62)

- Or $K(\Omega_n) > n - c$ and $\Omega_n$ can’t be compressed by more than a constant.

Universality

- We have a bound on the universal probability of a string $p_U(x)$ as follows:

$$p_U(x) = \sum_{p : U(p) = x} 2^{-\ell(p)} \geq 2^{-\ell(p^*)} = 2^{-K(x)}$$

(27.63)

where $p^* = \arg\min_{p : U(p) = x} \ell(p)$
- In fact, we have that:

$$2^{-K(x)} \leq p_U(x) \leq c 2^{-K(x)}$$

(27.64)

- Or

$$K(x) - c' \leq \log \frac{1}{p_U(x)} \leq K(x)$$

(27.65)

so that $\frac{1}{p_U(x)}$ is the Shannon “information” of $x$.
- So, Shannon lengths are as natural as $K(x)$ when using $p_U(x)$ as a distribution.
Also, the entropy of $p_U(x)$ is $E_K(X)$ where $X$ is a random variable drawn via $p_U$.

So, we can think of this as the universal entropy or the universal complexity of the universe.

All, assuming, the Occam’s razor assumption as being the guiding principle of the universe.