Outline of Lecture 2

Kolmogorov Complexity

- Plain complexity and the invariance theorem.
- Basic properties of C.
- Incompressibility and randomness oscillations.
- Prefix-free complexity K.
- Schnorr's Theorem.
- The Ample Excess Lemma.
- Chaitin's Ω.

Machine complexity

Let M be a Turing machine. M computes a partial recursive function $2^{<\mathbb{N}} \to 2^{<\mathbb{N}}$.

We define the M-complexity of a string x as

$$C_M(x) = \min\{|\sigma| : M(\sigma) = x\}$$

where $\min \emptyset = \infty$.

The complexity of x depends on the choice of M. Can we choose M so that it reflects the "true" complexity of x?

A machine R is optimal if for every machine M there exists a constant e_M such that

$$(\forall x)\ [C_R(x) \le C_M(x) + e_M].$$

The Invariance Theorem

Theorem [Kolmogorov]

There exists an optimal machine R.

Proof.

- Let (M_e) be an effective enumeration of all Turing machines.
- On input σ, R parses σ and finds unique e and τ such that $\sigma = 0^e 1\tau$. Then R outputs

$$R(0^e 1\tau) = M_e(\tau),$$

  i.e. R is essentially a universal Turing machine.
- It is now easy to see that for all e,

$$(\forall x)\ [C_R(x) \leq C_M(x) + e_M + 1].$$

Kolmogorov Complexity

We define the Kolmogorov complexity of a string x as

$$C(x) = C_R(x)$$

By the invariance theorem, any other machine complexity will "undercut" C by at most a constant.

If σ is an M_e-program for x, then $0^e1\sigma$ is an R-program for x.

Basic Properties of C

There exists an e such that for all x, $C(x) \leq |x| + e$.

- e is the index of a copying machine that just outputs the input. Obviously, x is an M_e-program for x.

For each length n, there exist incompressible strings of length n, i.e. strings x with $C(x) \geq |x|$.

- There are $\sum_{k=0}^{n-1} 2^k = 2^n - 1$ programs of length $< n$.

C cannot be increased by computable transformations.

- If $f : 2^{<\mathbb{N}} \to 2^{<\mathbb{N}}$ is (partial) computable, then there exists a c such that for all x such that $f(x)\downarrow$, $C(f(x)) \leq C(x) + c$.

Algorithmic Properties of C

C is not computable.

- The set $D = \{x \colon C(x) < |x|\}$ is simple – r.e. and the complement is infinite but does not contain an infinite r.e. subset.
- Assume the complement of D contains an infinite r.e. set. Then it also contains an infinite computable set $Z = \{z_1 < z_2 < \ldots\}$.
- A program for z_i is given by the index of the machine computing Z together with the index i, which can be coded by $\log i$ bits. Hence $C(z_i) \leq \log i + c$.
- For large enough i this contradicts that z_i is incompressible.
- Simple sets cannot be computable since this would mean the set and its complement are r.e.
- If C were computable, so would be D.

Algorithmic Properties of C

The noncomputability of C limits its use for practical purposes.

Possible remedies:

- Allow only a fixed number of steps for "decompression". Formally, let g be a total recursive function with $g(n) \geq n$. Define the time-bounded complexity

$$C^g(x) = \min\{|\sigma| : R(\sigma) = x \text{ in at most } g(|x|) \text{ steps}\}.$$

- Replace R by a computable compression/decompression mechanism (like any general compression algorithm – gzip etc.).

Algorithmic Properties of C

However, C is right-enumerable or enumerable from above:

- There exists a computable function $g : 2^{<\mathbb{N}} \times \mathbb{N} \to \mathbb{N} \cup \{\infty\}$ such that for all x, s, $g(s+1, x) \leq g(s, x)$ and

  $$\lim_s g(s, x) = C(x).$$

  For instance, we can take

  $$C_s(x) = \min\{|\sigma| \colon R(\sigma) = x \text{ in at most } s \text{ steps}\}$$

- Equivalently, the set

  $$\{(x, m) \colon \ C(x) < m\}$$

  is recursively enumerable.

Machine-independent Characterization of C

A function $D : 2^{<\mathbb{N}} \to \mathbb{N}$ satisfies the counting condition if $\{x\colon D(x) < k\} < 2^k$ for each k.

- The counting argument above shows that every machine complexity C_M satisfies the counting condition.

Proposition

If D is right-computable and satisfies the counting condition, then there exists a machine M such that for all x, $C_M(x) = D(x) + c$.

- It follows that C is given as a minimal (with respect to pointwise domination within a constant) right-computable function satisfying the counting condition.

Randomness as Incompressibility (I)

Conjecture: A sequence X is ML-random iff all of its initial segments are incompressible, i.e. iff for some constant c,

$$(\forall n)\ [C(X\restriction_n) \geq n - c]$$

Unfortunately, this is not true of any infinite sequence.

Theorem [Martin-Löf]

Let $k \in \mathbb{N}$. For any sufficiently long string x there exists an initial segment $y \subseteq x$ such that $C(y) < |y| - k$.

Randomness as Incompressibility (I)

Proof

- Let z be an initial segment of x.
- Let $n = n(z)$ be the index of z in a standard length-lexicographical ordering/enumeration of $2^{<\mathbb{N}}$.
- Let y be the length n extension of z along x, i.e. $y = z\sigma \subseteq x$ and $|\sigma| = n$.
- There is a machine that, given σ as input, outputs $z\sigma$.
- Hence $C(y) \leq |\sigma| + c$, where c is independent of y.
- On the other hand, $|y| = |z| + |\sigma|$, so if we choose z such that $|z| > k + c$, it follows that $C(y) < |y| - k$.

Failure of Subadditivity

The complexity of a concatenation can be higher than the complexities of its parts.

Given strings x, y, we should be able to combine programs for them to obtain a program for $z = xy$.

Hence it should be true that $C(xy) \le C(x) + C(y) + c$.

The problem is that, given a concatenation of descriptions for x and y, respectively, we cannot tell where the description of x ends and that of y begins.

Failure of Subadditivity

Corollary

Let $k \in \mathbb{N}$. There exists an x such that for some splitting $x = yz$ we have $C(x) > C(y) + C(z) + k$.

Proof

- Let c be such that $C(x) \leq |x| + c$ (c is the index of the copying machine).
- Pick an incompressible, sufficiently long x, $C(x) \geq |x|$.
- Let $l = k + c$ and use the preceding theorem to find an initial segment $y \subseteq x$ such that $C(y) < |y| - l$.
- Then for z such that $x = yz$, we have

$$C(y) + C(z) + k < |y| - k - c + |z| + c + k = |x| \leq C(x).$$

Randomness Oscillations

One can analyze these phenomena further to get an assessment on how incompressibility for C can fail along an infinite sequence.

Theorem [Martin-Löf]

Let $f : \mathbb{N} \to \mathbb{N}$ be a total computable function such that $\sum_n 2^{-f(n)} = \infty$. Then, for any sequence X, there exist infinitely many n such that

$$C(X \upharpoonright_n) \leq n - f(n).$$

For example, we can choose $f(n) = \log n$.

A “Better” Version of C?

One of the intended meanings of Kolmogorov complexity is information theoretic:

> If σ is a “minimal” program for x, σ contains precisely the information necessary to produce x.

But a string σ does not only contain its bits as information, it contains also its length.

> *This was used in the previous results.*

We should therefore somehow “incorporate” the length of a program into the definition of complexity.

A “Better” Version of C?

From a different perspective:

> The failure of subadditivity is due to the fact that we cannot, if we concatenate two descriptions, effectively tell where one ends and the other begins.

Instead of using $\sigma\tau$, we could use $0^{|\sigma|}1\sigma\tau$.

> $0^{|\sigma|}1\sigma$ is called a self-delimiting description of σ.

We will define a version of complexity that allows only self-delimiting descriptions.

Prefix-free Sets

Definition

A set $W \subseteq 2^{<\mathbb{N}}$ is prefix-free if for any $x, y \in W$,

$$x \subseteq y \quad \text{implies} \quad x = y.$$

In other words, no two elements of W are prefixes of one another.

Order theoretic:

> W is an antichain with respect to the partial order $\subseteq$ of strings.

Example: Phone numbers.

Prefix-free Kolmogorov complexity

A machine M is prefix-free if its domain is a prefix-free set. A prefix-free machine S is optimal if for every prefix-free machine M, $C_S \leq C_M + c$.

Proposition

There exists an optimal prefix-free machine S.

Proof:

- Enumerate all Turing machines.
- Whenever we see that some machine M_e is not prefix-free, we stop enumerating its domain. This way we convert it to a prefix-free machine $\tilde{M}_e$. If M_e is already prefix free, it remains unaltered.
- If $(\tilde{M}_e)$ is an enumeration of all (and only) prefix-free machines, we define $S(0^e 1\sigma) = \tilde{M}_e(\sigma)$.

Prefix-free Kolmogorov complexity

Definition

The prefix-free complexity of a string x is defined as

$$K(x) = C_S(x).$$

Properties of K

Algorithmic properties

- K is not computable.
- K is enumerable from above.

Upper bounds are harder than for C

- The copying machine is not prefix-free.
- We can replace it by the machine $M(0^{|x|}1x) = x$.
- This yields $K(x) \leq^+ 2|x|$. ($\leq^+$ means "$\leq \cdots + c$")
- General idea: Code x by x + self-delimiting code for $|x|$.
- The shortest self-del. code for $|x|$ is given by a program of length $K(|x|)$.
- Hence $K(x) \leq^+ |x| + K(|x|) \leq^+ |x| + 2\log|x|$.

Relating K and C

Proposition

$K(x) \leq^{+} K(C(x)) + C(x)$.

Proof

- Define machine M: On input τ search for decomposition $\tau = \sigma\eta$ such that $S(\sigma) \downarrow = k$, $k = |\eta|$. (S is the universal prefix-free machine.)
- If such decomposition is found, M simulates $R(\eta)$. (R is the universal machine for C.)
- M is prefix free.
- If η is a shortest R-description of x and σ is a shortest S-description of $|\eta|$, then M outputs x.
- Hence $K(x) \leq^{+} |\sigma| + |\eta| = K(C(x)) + C(x)$.

Relating K and C

Corollary

$C(x) \leq^+ K(x) \leq^+ C(x) + 2\log C(x) \leq^+ C(x) + 2\log(|x|)$.

We can also get a first "approximation" to subadditivity.

$$C(xy) \leq^+ K(x) + C(y).$$

- Search for decomposition of input into S-program for x and R-program for y.

Randomness as Incompressibility (II)

Proposition

The sequence $W_n = \{\sigma \colon K(\sigma) \leq |\sigma| - n\}$ is a ML-test.

Proof

- The W_n are uniformly r.e. since K is enumerable from above.
- Observation: If $V \subseteq 2^{<\mathbb{N}}$ is prefix-free, then $\sum_{\sigma \in W} 2^{-|\sigma|} \leq 1$.
- Each of the σ in W_n has a program τ of length $\leq |\sigma| - n$.
- These τ form a prefix-free set V_n.
- Hence $\sum_{\sigma \in W_n} 2^{-|\sigma|} \leq \sum_{\tau \in V_n} 2^{-(|\tau|+n)} \leq 2^{-n}$.

Randomness as Incompressibility (II)

It follows that if X is ML-random, it will pass the test (W_n).

This means that from some level c on (the W_n are nested), X is not covered by W_n for $n > c$.

This in turn means that

$$(\forall n)\ [K(X\restriction_n) \geq n - c].$$

In other words, if X is ML-random, its initial segments are incompressible with respect to K.

Randomness as Incompressibility (II)

Can we prove a converse of this? If the initial segments of X are incompressible, does it follow that X is random?

We want to show that if we have a ML-test, we can use it to compress initial segments that are covered by it.

For this, we will study a new way of devising prefix-free machines.

- This will at the same time give a new characterization of K.

Discrete Semimeasures

Definition

A discrete semimeasure is a function $m : 2^{<\mathbb{N}} \to [0, 1]$ such that

$$\sum_{x \in 2^{<\mathbb{N}}} m(x) \leq 1$$

Think of a semimeasure as an incomplete probability distribution over $2^{<\mathbb{N}}$ (or equivalently, $\mathbb{N}$).

A semimeasure m is called optimal for a family $\mathcal{F}$ of semimeasures if $m \in \mathcal{F}$ and it multiplicatively dominates all semimeasures in $\mathcal{F}$, i.e. if

$$(\forall f \in \mathcal{F})\ (\exists c_f)\ (\forall x)\ [f(x) \leq c_f m(x)].$$

Discrete Semimeasures

Theorem [Levin]

There exists a semimeasure $\widetilde{m}$ that is optimal for the family of left-computable discrete semimeasures.

One can construct such a semimeasure along the lines of the previous universality constructions.

But we will actually see that the function

$$\widetilde{m}(x) = 2^{-K(x)}$$

is an optimal semimeasure. This is known as the Coding Theorem.

The Coding Theorem

Theorem [Levin]

If $\widetilde{m}$ is an optimal left-computable semimeasure, then $-\log \widetilde{m} =^{+} K$.

Proof

- It suffices to show that 2^{-K} is an optimal left-computable semimeasure.
- 2^{-K} is left-computable, since K is enumerable from above.
- Let m be a left-computable semimeasure. We construct a prefix-free machine M such that $K_M(x) \leq^{+} -\log m(x)$.

The Coding Theorem

Proof

- Let $\{(x_t, k_t) \colon t = 1, 2, \ldots\}$ be an enumeration of the set $\{(x, k) \colon 2^{-k} < m(x)\}$ without repetition.
- Then $\sum_t 2^{-k_t} = \sum_x \sum_t \{2^{-k_t} \colon x_t = x\} \leq \sum_x 2m(x) < 2$.
- Cut off adjacent intervals I_t of length $2^{-k_t - 1}$ from the left side of $[0, 1]$.
- If $[\![\tau]\!]$ is the largest binary subinterval for some I_t, let $M(\tau) = x_t$. Otherwise let M be undefined.
- M is obviously prefix-free and partial recursive.
- It follows from the enumeration that for all x exists a t such that $x_t = x$ and $m(x)/2 < 2^{-k_t}$.
- Hence for every x there exists a τ such that $M(\tau) = x$ and $|\tau| \leq -\log m(x) + 4$.

The Kraft-Chaitin Theorem

The Coding Theorem gives us a useful methods to prove complexity bounds.

Corollary

Suppose we have a computable sequence of "requests" of the form (r_i, x_i), meaning that we want to build a prefix-free machine M such that for all i exists σ_i with $|\sigma_i| = r_i + c$ and $M(\sigma_i) = x_i$. Such a machine exists iff the function $m(x_i) = 2^{-r_i}$ is a semimeasure.

The proof is analogous to the construction in the previous proof.

Randomness as Incompressibility (III)

Now let (W_n) be a ML-test that covers X.

Define $m_n(\sigma) = n2^{-|\sigma|}$ if $\sigma \in W_n$ (0 otherwise), and $m = \sum_n m_n$.

m is enumerable from below.

$\sum_\sigma m(\sigma) \leq \sum n/2^n < \infty$.

Deleting finitely many strings from W does not change the covering properties of the test and turns m into a semimeasure.

Hence for some c, $m \leq c\,2^{-K}$.

Randomness as Incompressibility (III)

Given n there exists l_n such that $X\restriction_{l_n} \in W_n$.

Hence $m_n(X\restriction_{l_n}) = n2^{-l_n}$, which implies

$$n = \frac{m_n(X\restriction_{l_n})}{2^{-l_n}} \leq \frac{m(X\restriction_{l_n})}{2^{-l_n}} \leq \frac{2^{-K(X\restriction_{l_n})}}{2^{-l_n}}.$$

This yields

$$\limsup_n \frac{2^{-K(X\restriction_{l_n})}}{2^{-l_n}} = \infty,$$

or equivalently

$$(\forall n)\,(\exists l_n)\,[K(X\restriction_{l_n}) < l_n - n].$$

Schnorr's Theorem

We have proved the second main theorem of algorithmic randomness, better known as Schnorr's Theorem.

Theorem

A sequence is ML-random iff there exists a c such that for all n,

$$K(X\restriction_n) \geq n - c.$$

The Ample Excess Lemma

For a random sequence, the distance between $K(X\restriction_n)$ and n must in fact go to infinity.

Theorem [Miller and Yu]

X is ML-random iff $\sum_n 2^{n-K(X\restriction_n)} < \infty$.

Proof: $(\Leftarrow)$
If X is not ML-random, then there exist infinitely many n such that $K(X\restriction_n < n$, which implies

$$\sum_n 2^{n-K(X\restriction_n)} = \infty.$$

The Ample Excess Lemma

Proof: $(\Rightarrow)$

- Fix a length m. Let's count the total 'gaps' along strings of length n:

$$\sum_{|\sigma|=m} \sum_{n \leq m} 2^{n-K(\sigma\upharpoonright_n)} = \sum_{|\sigma|=m} \sum_{\tau \subseteq \sigma} 2^{|\tau|-K(\tau)} = \sum_{|\tau| \leq m} 2^{m-|\tau|} 2^{|\tau|-K(\tau)}$$

$$= 2^m \sum_{|\tau| \leq m} 2^{-K(\tau)} < 2^m$$

- Hence at most 2^{m-c} strings σ of length m have

$$\sum_{n \leq m} 2^{n-K(\sigma\upharpoonright_n)} \geq 2^c.$$

- Therefore, $\lambda\{Y\colon \sum_{n \leq m} 2^{n-K(Y\upharpoonright_n)} \geq 2^c\} \leq 2^{-c}$.
- And thus, $U_c = \{Y\colon \sum_n 2^{n-K(Y\upharpoonright_n)} \geq 2^c\}$ has measure at most 2^{-c}. (U_c) forms a test that covers all Y for which $\sum_n 2^{n-K(Y\upharpoonright_n)} = \infty$.

Chaitin's Ω

While there is an abundance of random sequences, it is hard to come up with a distinguished example.

Chaitin defined the real number

$$\Omega = \sum_{\sigma \in \mathrm{dom}(S)} 2^{-|\sigma|}.$$

Theorem [Chaitin]

The binary expansion of Ω is a ML-random sequence.

Chaitin's Ω

Proof

- We build a (plain) machine M.
- On input x of length n, wait for t such that $0.x \leq \Omega_t < 0.x + 2^{-n}$, where

$$\Omega_t = \sum_{S(\sigma)\downarrow \text{ in at most } t \text{ steps}, \, |\sigma| \leq t} 2^{-|\sigma|},$$

  the approximation to Ω at stage t.
- If such t is found, output the least string y not in the range of S_t
- If $x = \Omega \upharpoonright_n$, then such t exists.
- By stage t all S-descriptions of length $\leq n$ have appeared, otherwise $\Omega > \Omega_t + 2^{-n}$.
- Thus $M(x) = y$ and $K(y) > n$.
- Hence $K(\Omega \upharpoonright_n) \geq^+ K(M(\Omega \upharpoonright_n)) > n$.

Digression: Clustering via Information Distance

Given two strings x, y, let $\langle x, y \rangle$ be a standard pairing function, for example $\langle x, y \rangle = 0^{|x|}1xy$.

- Think of a pairing function as a way to code x, y, and a way to tell them apart.

Definition

Define the information distance between two strings x, y as

$$E(x, y) = K(\langle x, y \rangle) - \min\{K(x), K(y)\}.$$

E minorizes (up to a constant) all computable, nonnegative, symmetric functions between strings.

- This means if x, y are close with respect to some distance function D, they will also be close with respect to E.

Digression: Clustering via Information Distance

Since information distance should be measured relative to length, we define the normalized information distance

$$\mathrm{NID}(x, y) = \frac{K(\langle x, y\rangle) - \min\{K(x), K(y)\}}{\max\{K(x), K(y)\}}.$$

For practical purposes, replace K by C_M with total computable prefix-free compressor/decompressor (e.g. gzip).

The Coding Theorem lets us replace a prefix-free compressor by any enumerable semimeasure.

Digression: Clustering via Information Distance

Google probability

Let $\mathcal{S}$ be the set of all Google search terms.

Let $\mathcal{W}$ be the set of all web pages indexed ($\sim 10^{10}$).

Google probability of a search term x:

- Let $\mathbf{x}$ denote all pages on which x appears.
- $L(x) = |\mathbf{x}|/|\mathcal{W}|$.

Problem: L is not a semimeasure (events overlap).

Modify counting: $N = \sum_{\{x,y\} \subseteq \mathcal{S}} |\mathbf{x} \wedge \mathbf{y}|$.

Set $g(x, y) = |\mathbf{x} \wedge \mathbf{y}|/N$. Then $\sum_{\{x,y\} \subseteq \mathcal{S}} g(x, y) = 1$, hence we can derive a prefix-free complexity, the Google complexity G.

Digression: Clustering via Information Distance

Google distance

Set $g(x,y) = |\mathbf{x} \wedge \mathbf{y}|/N$. Then $\sum_{\{x,y\} \subseteq \mathcal{S}} g(x,y) = 1$, hence we can derive a prefix-free complexity, the Google complexity G.

Based on this, define the normalized Google distance

$$\mathrm{NGD}(x,y) = \frac{G(\langle x,y\rangle) - \min\{G(x), G(y)\}}{\max\{G(x), G(y)\}}.$$

Application: Clustering using "Google semantics"