Lecture 9. Resource bounded KC

- K-, and C- complexities depend on unlimited computational resources. Kolmogorov himself first observed that we can put resource bounds on such computations. This was subsequently studied by Barzdins, Loveland, Daley, Levin, Adleman.
- Hartmanis, Ko, Sipser started a new trend in computer science in the 1980's.
- In the 1960's, two parallel theories were developed:
  - Computational complexity – Hartmanis and Stearns, Rabin, Blum, measuring time/space complexity
  - Kolmogorov complexity, measuring information.
- Resource bounded KC links the two theories.

Theory (I will do C, same applies for K)

- $C^{t,s}(x|y)$ is the t-time s-space limited Kolmogorov complexity of x condition on y. I.e. the length of shortest program that with input y, *produces* x in time t(n) and space s(n) with n = |x|.
- In standard C, K complexity, it does not matter if we say “produces x” or “accepts x”, they are the same. But in resource bounded case, they are likely to be different. So Sipser defined $CD^{t,s}(x|y)$ to be the length of the shortest program that with input y *accepts* x in time t(n) and space s(n).
- When we use just one parameter such as time, we will simply write C^t or CD^t.

Relationship between C^t and CD^t

Theorem (Sipser). (a) For any t, $CD^t(x) \leq C^{t+O(n)}(x)+O(1)$. (b) Reversely, for polynomial p, there is polynomial q, $C^q(x|A) \leq CD^p(x)+O(1)$, where A is an NP-complete set.

Remark: When t is exponential, C^t and CD^t are same.

Proof. (a) is trivial. (b) Define T such that it accepts only x and it runs in time p(n). Use T with oracle A to determine the successive bits of x. The set A is defined by $A=\{<T,y,1^t,1^n>$: T accepts yz in time t(n) for some z s.t. $|yz|=n\}$ is NP-complete. So we can find a string accepted by T by querying its successive bits (next bit would be 0 or 1, ask both ways) in time polynomial in p(n). QED

Hierachy

- (Hartmanis) It is possible to obtain a resource bounded hierarchy --- there are always strings that need more seed length, time, or space to compute. I.e. define C[f(n), t(n), s(n)] to be the set of strings of length n that can be generated by programs of length f(n), in time t(n) and space s(n). Then increasing each of the three parameters (to some degree) will result a bigger class.

Compression relative to a set

- If A is computable, $|x|=n$ and $x \in A$, we know:

  $$C(x|A),\ CD(x|A) \leq \log|A^{=n}|+O(\log n)$$

- But with time limit, this is not so easy. The above proof requires us to check all 2^n strings to see which are in A, even if A is in P. We will only provide some sample of the results in this direction.

Theorem [Buhrman-Fortnow-Laplante] Let A be *any set given as oracle*. There is a polynomial p s.t. for all $x \in A^{=n}$

$$CD^p(x|A^{=n}) \leq 2\log|A^{=n}|+2\log n+O(1).$$

Proof of Buhrman-Fortnow-Laplante's theorem

Proof.

Lemma. Let $S=\{x_1, \ldots, x_d\} \subseteq \{0, \ldots, 2^n-1\}$. For all $x_i \in S$, there exists a prime $p_i \leq 2dn$ such that for all $j \neq i$, $x_i \neq x_j \bmod p_i$

Proof of Lemma. For each pair (i,j) there are at most $\log_c 2^n = \log 2^n / \log c$ different primes p s.t. $c \leq p \leq 2c$, and $x_i = x_j \bmod p$. Fixing i, we have d-1 pairs (x_i, x_j) for $j \neq i$. Total there are at most $(d-1)\log_c 2^n$ such primes. By prime number theorem, there are c/logc primes in the range of [c,2c]. Taking $dn > c > (d-1)n$, then there must be at least one p_i s.t. for $j \neq i$, $x_i \neq x_j \bmod p_i$. Since $p_i \leq 2c$, $p_i \leq 2dn$. QED

Use the lemma with $d=|A^{=n}|$, for $x \in A^{=n}$, to get p_x. We can write a program: input y, if $y \notin A^{=n}$ by oracle query, then reject y; else check if $y=x \bmod p_x$ then accept y, else reject y.

The above program needs information: p_x and $x \bmod p_x$, $p_x \leq 2|A^{=n}|n$.

This is $2|p_x|+O(1)$, approximately $2\log|A^{=n}|+2\log n+O(1)$. QED

- Remark: it is possible to improve to $\log|A^{=n}|$ if we use an extra random string beside the oracle in the conditional (Sipser, Theorem 7.2.2 in textbook).

Using it …

Example. There is a recursive set A s.t. $P^A \neq NP^A$

Proof. Define $f(1)=2$, $f(k)=2^{f(k-1)}$. By diagonalization, choose B so that $B \subseteq \{1^{f(k)}\}$ and $B \in DTIME[n^{\log n}]-P$. Use B to construct A as follows: For each $1^{f(k)} \in B$, put first string of length $n=f(k)$ from

$C[\log n, n^{\log n}] - C[\log n, n^{\log\log n}]$ (*)

in A for each k.

A is recursive and $B \in NP^A$ as an NTM can guess the string in A in poly time. But B is not in P, and a DTM cannot find a string in A to query (in polynomial time) by (*). Hence the oracle A does not help it. Thus B is also not in P^A.

QED

- Remark: We have used the fact that some strings are hard to get to, without enough time.

Instance Complexity (Orponen-Ko-Schoning-Watanabe)

- This is a cute concept. Let T be a UTM. Given a set A and time bound t, define the instance complexity of x as: (with A(y)=1 if y in A and 0 otherwise)

  $ic_T^t(x: A) = \min\{|p|: T(p,x)$ halts, and for all y such that $T(p,y)$ halts, $T(p,y)=A(y)$ in time $t.\}$

- You can prove an invariance theorem for this also. So we will drop T. The goal is to identify hard instances that makes a language hard.
- Facts:
  - $CD^t(x)=ic^t(x: \{x\})$
  - A set A is in P iff there is a polynomial p, such that for all x, $ic^p(x: A)$=constant.

Levin's Kt complexity and universal search

- Imagine how much time God takes to create a sequence (such as our genomes). Assume He can flip a coin, and He has Turing machines. He can either keep on flipping a coin or enumerate until he gets x in time $2^{|x|}$, or He can start to compute x from a short seed.
- For random sequence x, it takes expected $2^{|x|}$ time. But for easy sequences, it can be generated from shorter (random) seed in time t. So let's define

  $$\text{age}(x)=\min_p \{2^{|p|}t : U(p)=x \text{ in } t \text{ steps}\}$$

- Then Kt(x)=log age(x).

Universal search

- If there is a polynomial time algorithm to solve NP-hard problems, can we find it?
- Yes, we now describe a method of Levin who called it "universal search".

Lemma. If there is an alg A that solves SAT in time t(n), then the following algorithm solves it in time ct(n) for some constant c.

Proof. The algorithm is: Simulate all TM's in the following dovetailing fashion: T_1 every 2^{nd} step; T_2 every 2^{nd} step among the remaining steps; T_3 every 2^{nd} step among the remaining steps; … Then if T_k solves the SAT problem in time t(n), then this algorithm solves it in $2^k t(n) + O(1)$ steps. QED

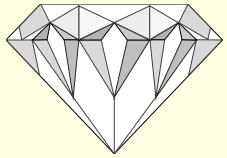
Remark: Levin's algorithm uses Kt complexity and is slightly faster.

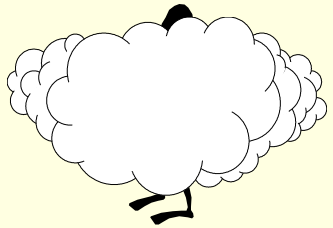
Logical depth (Chaitin, Bennett)

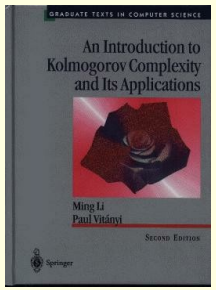
- We think a book in number theory is deep. It is not random, it has very low Kolmogorov complexity. In fact, all its theorems can be derived automatically by enumerating all the proofs --- hence $C(\text{number theory book})=O(1)$.
- Some other things are shallow: children books, sequence 1^n. In fact a Kolmogorov random sequence such as ideal gas is also shallow.
- Can we identify information with depth? 1^n does not contain information, but a random string also does not contain information.

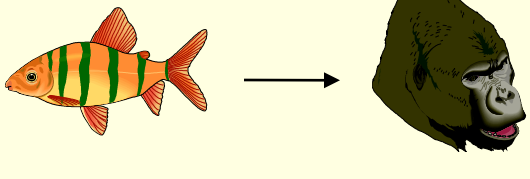
Shallow and deep things

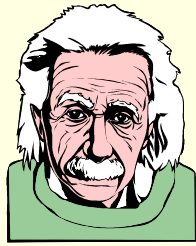
- **Crystal is shallow**
- **Gas is also shallow**
- **A math book is deep**
- **Life is deep**











A structure is deep, if it is superficially random but subtly redundant ... Bennett

Defining "depth"

- Attempt 1. We cannot use the number of steps needed to compute from x*, the shortest program generating x, to x, since perhaps there is another program slightly longer than x* but it runs much faster – such a definition is not "stable".
- Attempt 2. So we relax to almost minimum programs. We can say x has depth d within error 2^{-b} if x can be computed in d steps by a program p s.t. $|p| \leq |x^*|+b$, i.e. $2^{-|p|}/2^{-K(x)} \geq 2^{-b}$. But what about there are many such programs of length $|x^*|+b$? Consider them?
- Thus, we should consider all such programs

$$Q(x) = \Sigma_{U(p)=x} 2^{-|p|}$$

Defining "depth", cont.

Definition. The depth of a string x at significance level $\varepsilon=2^{-b}$ is

$$\text{depth}_\varepsilon(x)=\min\{t: Q^t(x)/Q(x) \geq \varepsilon\},$$

where $Q^t(x)=\Sigma_{p \text{ runs in } t \text{ steps}} 2^{-|p|}$. We say a string x is (b,d) deep if $d=\text{depth}_\varepsilon(x)$ and $\varepsilon=2^{-b}$.

Remark. Thus a (d,b)-deep string receives $1/2^b$ fraction of its algorithmic probability from programs running in d steps.

Examples

- Remember the halting probability Ω, and χ (its i-th bit indicates if the i-th TM halts). Although both encodes the halting information of TMs, χ is deep and Ω is shallow. This is because Ω encodes the halting problem in maximum density and it is recursively indistinguishable from a random string, hence practically useless. I.e. $K(\Omega_{1:n}) \geq n-O(1)$. $K(\chi_{1:n})$ on the other hand is small (logn). If t is the minimum time required to compute $\chi_{1:n}$ from a logn-sized program, then it is t-deep at all significance levels. In fact it is very deep as t grows faster than any computable function.
- Consider our genome. How deep is it? If it needs to evolve from nothing (or very shallow strings), how long does it take? This is possibly a computation in PSPACE (linear space). If so, and if P=PSPACE, then every string (genome) has polynomial depth. Otherwise, some may have exponential depth. We of course know (from the historical records) that our genomes did not have exponential depth, if God did not play dice with us.