Complexities

Péter Gács

Computer Science Department
Boston University

Spring 2018

Outline

- Models of computation: non-uniform and uniform
- Kolmogorov complexity, uncomputability
- Cost of computation: time, space
- NP-completeness
- Randomness
- Algorithmic probability
- Logical depth

Models of computation

- Logic circuit: A network whose nodes contain:
  - Logic gates (like AND, OR, NOT, NOR).
  - Inputs and outputs.
  - If the network is not acyclic, also some memory elements.

  A set of gates is universal if for every n and every Boolean function $f : \{0, 1\}^n \to \{0, 1\}$, there is a circuit built from such gates computing it. In quantum computing, this is frequently meant by computational universality.
- The cost of a circuit can be measured by its size, width, depth, working time, and so on.
- In the theory of computing, this computational model is not sufficiently expressive since it allows only a finite number of possible inputs. The notion of computability cannot even be formulated here.

- The approriate models of computation have an infinite amount of memory: Examples:
  - Turing machines
  - Cellular automata
  - Random access machine (don't ask the details).
  - Many others (including uniform circuits).

  All the reasonable models are equivalent in what functions they can compute.
- We can list all Turing machines, indexing them as T_p. A Turing machine U is universal if it interprets its input as a pair (p, x) where p is a program of an arbitrary Turing machine T_p and x is the input: so $U(p, x) = T_p(x)$.

Compression

Information in some 0-1 string

$$x = x_1 x_2 \dots x_n.$$

If $x = 0101 \dots 01$ then can be described by just saying: "take $n/2$ repetitions of 01". The sequence can be "compressed", or "encoded" into a much shorter string.
Fixing a standard for interpreting compressed descriptions: Some computer T reading the description p as input.

$$C_T(x) = \min_{T(p)=x} |p|.$$

Description complexity of x on T.

Invariance

There is an optimal machine U for descriptions: for every machine T there is a constant c with

$$C_U(x) < C_T(x) + c.$$

All the machines you are familiar with are optimal. So, the description complexity of a string x is essentially an inherent (and interesting) property of x. From now on,

$$C(x) = C_U(x).$$

Description complexity upper and lower bounds

Upper bound It is easy to see that $C(x) \leq |x| + c$ for some constant c.

Lower bound For each k the number of binary strings x of length n with $C(x) < n - k$ is at most 2^{n-k} (so most strings are nearly maximally complex). Indeed, the total number of strings with descriptions of length $< n - k$ is at most $1 + 2 + \cdots + 2^{n-k-1} < 2^{n-k}$.

The latter proof did not provide any concrete example of a string with even $C(x) > 100$. Not by accident.

- Description complexity is deeply uncomputable. Proof via an old paradox.
- There are some numbers that can be defined with a few words: say, "the first number that begins with 100 9's", etc. There is a first number that cannot be defined by a sentence shorter than 100. But—I have just defined it!
- This is a paradox, exposing the need to define the notion of "define". Now, let "p defines x" mean $U(p) = x$.

- Assume $C(x)$ is computable, so there is an algorithm that on input x, computes $C(x)$. Then there is also an algorithm Q that on input k, outputs the first string $x(k)$ with $C(x) > k$.
- Let q be the length of a program on U for the above algorithm Q. For some number k, we can write now some program $r(k)$ for U that outputs $x(k)$.
- We also need some constant p bits to tell U what to do with this information, but then

$$|r(k)| \leq p + q + \log_2 k.$$

If k is sufficently large then this is less than k: contradiction.

Cost of computation

- Given a universal Turing machine U,

$$\mathrm{time}_U(p, x)$$

  is the number of steps of $U(p, x)$. Could be viewed as the cost of this computation.
- This notion seems too dependent on arbitrary choices.
  - Depends on the machine model used. “Random access machine” may do it faster than a Turing machine.
  - Why not measure memory (storage, space) used instead?
- Fortunately, any two “reasonable” computation models (no massive parallelism), say Turing machines and cellular automata, simulate each other in polynomial time; so the dependence on the model is limited. (The exclusion of quantum computers is debatable!)
- There are some easy bounds between space and time cost, but the deeper relation between them is little understood.

- For an algorithm (a program) p on Turing machine U, its time complexity is defined in a worst-case manner:

$$t_p(n) = \max_{|x|=n} \mathrm{time}_U(p, x).$$

For example we say that it runs in time $O(n^2)$ if there are constants c, d with $t_p(n) \le cn^2 + d$.

- For technical reasons, though we can say whether a function $f(\cdot)$ is computable, we don't define its computational cost. Instead, we define complexity classes. We say that

$$f(\cdot) \in \mathrm{DTIME}(t(n))$$

if there is an algorithm computing $f(\cdot)$ in time $O(t(n))$.

- $P = \bigcup_k \mathrm{DTIME}(n^k)$ is the class of functions computable in polynomial time,
  $\mathrm{EXP} = \bigcup_k \mathrm{DTIME}(2^{kn})$ is the class of functions computable in exponential time.
- Let divide$(x, y) = 1$ if integer y (written in binary) divides integer x, and 0 otherwise.
  Let factorize$(x, y) = 1$ if x has some divisor $\leq y$ and 0 otherwise.
- There is a well-known polynomial algorithm for computing divide(x, y): we learned it in school.
  There is no known polynomial algorithm for computing factorize(x, y): the trial division algorithm is exponential.
- The biggest unsolved problems of computational complexity theory concern lower bounds. For example the most used cryptography algorithms use the unproved assumption that factorize$(\cdot, \cdot) \notin P$.

- The class P is very important for complexity theorists; typicaly, by an efficient algorithm, one means a polynomial-time one.
- Polynomial time algorithms are often contrasted with exponential-time ones. Consider the following two problems, both about a graph G of n vertices.
  - Find the largest number of disjoint edges.
  - Find the largest number of independent vertices.

  Brute-force search (trying all possibilities) solves both of these problems in exponential time, so both are in EXP.
- The first problem also has a (nontrivial) polynomial-time algorithm, so it is in P.

  The second problem is not known to have one, and since it is NP-hard (see later) most bets are against it.

Most spectacular results of computer science are positive: upper bounds on complexity, even even when they started as answers for questions on lower bounds.

Example In the 1950's Kolmogorov asked his students to prove that multiplication of two n-digit numbers takes n^2 elementary steps, just like the school algorithm.
The answer—with repeated improvements—was an upper bound $O(n \log n \log \log n)$.

- A simple diagonal argument, going back to Cantor and Gödel, shows that the partial function $U(x, x)$ computed by a universal Turing machine cannot be extended to a computable one.
- Let $H(x) = 1$ if $U(x, x)$ is defined (if $U(x, x)$ halts), and 0 if it is not. Finding the value of $H(x)$ is the famous halting problem: it is also undecidable.
- Let $H^t(x)$ be the same thing, after t steps. The same kind of diagonalization shows that

$$f(x) = H^{2^{|x|}}(x)$$

cannot be computed in time $2^{|x|}/|x|$, so

$$f(\cdot) \in \mathrm{DTIME}(2^n) \setminus \mathrm{DTIME}(2^n/n).$$

Reductions

Most undecidability results and lower bounds are proved via reduction. Consider an equation of the form

$$x^3 = 3y^6 - 2x^4 - x^2y + 11,$$

asking for integer solution. Hilbert's 10th problem about Diophantine equations asks for an algorithm to solve all such problems. Now we know that there is no such algorithm.
Let $D(E) = 1$ if Diophantine equation E is solvable, and 0 otherwise. A famous construction defines a computable function $\rho(x)$ with

$$D(\rho(x)) = H(x).$$

(ρ encodes the work of a universal Turing machine into equations.)
This shows that D is at least as hard as H, and we write

$$H \leq D.$$

- Generously considering all polynomial algorithms efficient, computer scientists are interested in polynomial-time reductions. If $f(x) = g(\rho(x))$ by a polynomial-time function $\rho(x)$, then we write

$$f \leq_p g.$$

  This upper-bounds the complexity of f but is used even more frequently to lower-bound the complexity of g.
- Function f is hard for a class of functions $\mathcal{C}$ (in terms of polynomial reductions) if $f \geq_p g$ for all elements of $\mathcal{C}$.
- f is complete for $\mathcal{C}$ if it is hard for $\mathcal{C}$ and also belongs to $\mathcal{C}$. So f is one of the hardest elements of $\mathcal{C}$.
- Example: the function $H^{2^{|x|}}(x)$ is complete for EXP.

Example Generalize the game of Go, to an $n \times n$ board.

- Let $W(x)$ be the function that is 1 if configuration x (an $n \times n$ matrix) is winning for White and 0 if it is not. A clever reduction shows that W is complete for EXP. So W can only be computed in exponential time.
- Let $W'(x)$ be 1 if White will win in $\leq n^2$ steps and 0 otherwise. A reduction shows that W' is complete for PSPACE, the class of functions computable using a polynomial amount of memory. What does this say about the time needed to compute $W'(x)$? Nothing, (other than bets). See below.

NP problems

- A subset of PSPACE holds particular interest: yes/no questions in which the "yes" answer (return value 1) has a proof checkable in polynomial time.
  Example: given a graph G of size n, let $I(G) = 1$ if G has an independent subset of size $n/2$ and 0 otherwise.
- The class of such functions (predicates) is called NP (for "nondeterministic polynomial", ignore why). An immense number of interesting and important problems belong to NP.
- $I(\cdot)$ is proved to be NP-complete. Does this lower-bound its time complexity? We don't know. In the inclusions below, we don't know which one is equality—just that all cannot be.

$$P \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}.$$

Still, the NP-completeness of a problem is considered a strong evidence for its hardness.

The following variant of Kolmogorov complexity is very convenient.
Let a Turing machine T be said to have the prefix property if whenever binary string p is a prefix of q and $T(p)$ is defined then $T(p) = T(q)$. For such a machine T let

$$K_T(x) = \min_{T(p)=x} |p|.$$

Again, there is an optimal prefix machine V, and we will write $K(x) = K_V(x)$. It is not hard to see that

$$C(x) \le K(x) \le C(x) + 2 \log C(x).$$

Let $P(x)$ be any computable probability distribution over finite strings x. The complexity upper and lower bounds generalize nicely: We have

$$K(x) \leq -\log P(x) + c_P.$$

for some constant c_P. On the other hand,

$$P\{\, x : K(x) < -\log P(x) - k \,\} \leq 2^{-k}.$$

This, with other considerations, justifies calling

$$d(x, P) = -\log P(x) - K(x)$$

the deficiency of randomness of x with respect to distribution P. We consider x more random when $K(x)$ is closer to its upper bound $-\log P(x) + c_P$.

Complexity and entropy

Let $H(P) = \sum_x P(x) \log(1/P(x))$ be the entropy of the computable distribution P. We have

$$|H(P) - \sum_x P(x)K(x)| \leq c_P$$

for a constant c_P. So entropy is nearly average complexity, justifiying the name "algorithmic entropy" for $K(x)$.

Algorithmic probability

Let us feed an infinite string of random bits π to our optimal prefix machine V. We write $V(\pi) = x$ if V halts on some prefix of π and outputs x. The algorithmic probability of x is defined as

$$\mathbf{m}(x) = \mathrm{Prob}\{V(\pi) = x\}.$$

This is the probability that the optimal prefix machine with a monkey at the terminal outputs x. The distribution $\mathbf{m}(x)$ is not computable (and does not add up to 1). It dominates all computable distributions: for every computable distribution P there is a constant d_P with

$$P(x) \leq d_P \cdot \mathbf{m}(x)$$

An important theorem says

$$K(x) = -\log \mathbf{m}(x) + O(1).$$

- Let $\mathbf{m}_t(x)$ be the probability that $V(\pi)$ outputs x in $\le t$ steps. The quantity

$$\mathrm{depth}_\varepsilon(x) = \min\{\, t : \mathbf{m}_t(x)/\mathbf{m}(x) \ge \varepsilon \,\}.$$

  is (a version of) Bennett's logical depth. It is larger than t if the conditional probability that x arises in t steps provided it arises at all is $\le \varepsilon$.
- Any simple random process ("randomized computation") needs at least $\mathrm{depth}_\varepsilon(x)$ steps to produce x with probability $\ge \varepsilon\mathbf{m}(x)$. So depth is a certain pedigree of long evolution (alas, uncomputable).
- If a string x is random with respect to a computable distribution P then its depth is nearly bounded by the time needed to sample P; so random strings are shallow (these include the simple strings, too).

- A variant (presented by Charlie) considered rather the difference

$$K^t(x) - K(x)$$

instead of $-\log \mathbf{m}_t(x)/\mathbf{m}(x)$ (for technical reasons, these are not quite the same).

- Little is known about the existence of strings of a certain depth. For large n, are there strings x of length n with, say,

$$K^{n^3}(x) \leq n/4, \quad K^{n^2}(x) > n/2?$$

The question $K^{n^2}(x) \leq n/2$ is of type NP.
We can produce a string x with $K^{n^2}(x) > n/2$ by brute force search, in time $n^2 2^n$. But who knows whether we can faster, say in time n^3?

Topics missed

- Randomized computing.
- Pseudo-randomness, cryptography.
- Can randomness be replaced with pseudo?
- Interactive proofs, IP = PSPACE.
- Transparent (holographic) proofs, their use to lowerbound the complexity of approximations.
- Quantum computing…