The theory of NP-completeness has been applied to explain why brute-force search is essentially unavoidable for many interesting problems. The 1979 book *Computers and Intractability: A guide to the theory of NP-completeness* by Garey and Johnson contains over a hundred examples. It has been extended to handle other types of problems beyond search, such as counting problems, problems in the so-called "polynomial-time hierarchy", as well as approximate optimization problems, which we will see later in the class.

One drawback of this theory, however, is that it measures computational hardness in terms of the worst-case running time of algorithms over all inputs of a given size. Sometimes it is more relevant how algorithms behave on "typical" inputs. Average-case complexity assumes that inputs come from some probability distribution and algorithms are allowed to err with some small probability over the choice of input. Two types of algorithms that fit nicely into this setting are learning algorithms (whose inputs consist of training examples, which are sometimes modeled as independent samples from some distribution) and cryptographic adversaries (whose success is determined by how likely they are to break a random execution of the system).

1 Distributional problems

A distributional problem is a computational problem together with an ensemble $\mathcal{D} = \{D_1, D_2, \dots\}$ of distributions on instances supported over finite subsets of $\{0,1\}^*$. An instance of "size" n is obtained by sampling from the distribution D_n. (Instances in the support of of D_n need not have length n, and in fact they do not even all need to have the same length, but in line of our notions of efficiency the length of such instances will usually be at most a polynomial in n.) Let us now give some examples that motivate the definitions of search and decision algorithms for NP-type distributional problems.

For example, D_n could be the following distribution on 3CNFs ϕ with n variables: Choose $m = n$ clauses independently, where each clause is chosen uniformly at random among all $8\binom{n}{3}$ possibilities that yield three distinct literals. It is known that formulas from D_n are almost always satisfiable as n grows, so the decision version of 3SAT under distribution $\mathcal{D}$ appears to be trivial to solve by an algorithm that always outputs 1. The search version, however, is more interesting as it is not clear how a satisfying assignment can be found. It turns out that there is an efficient algorithm that finds a satisfying assignment for almost all instances of D_n.[1]

We would like a search algorithm A for a distributional search problem $(S, \mathcal{D})$ to behave like this: Find a solution for most instances sample from the distribution, namely that when x is sampled from D_n, the probability that $A(x)$ is a solution for x under S is large for all n. But what if the distribution $\mathcal{D}$ places significant probability on instances that have no solution? If we modify the parameters in the above example so that $m = 10n$ we are exactly in this situation: The instances ϕ have more constraints and most of them now become unsatisfiable. Since the search algorithm can do nothing with such instances, we have to count them towards its success probability as well.

Definition 1. A *search heuristic* for a distributional search problem $(S, \mathcal{D})$ with error $\varepsilon(n)$ is an algorithm A such that for all n,

$$\Pr_{x \sim D_n}[x \text{ has a solution and } A(x) \text{ is not a solution for } x] < \varepsilon(n).$$

[1] The proof that instances of D_n are almost always satisfiable in fact works by showing that this algorithm works with high probability over the choice of ϕ.

Let us now turn to decision problems. As adding more clauses to our random 3CNF instances only decreases the probability that a random instance is satisfiable, for every n there should be some value of m for which the probability ϕ is satisfiable is close to $1/2$.[2] A random instance then has about the same likelihood of being satisfiable and unsatisfiable. The algorithm has to predict which and be correct most of the time.

Definition 2. A *decision heuristic* for a distributional decision problem $(f, \mathcal{D})$, $f\colon \{0,1\}^n \to \{0,1\}$, with error $\varepsilon(n)$ is an algorithm A such that n,

$$\Pr_{x \sim D_n}[A(x) \neq f(x)] < \varepsilon(n).$$

Now let us go back to our search algorithm for $(3\text{SAT}, \mathcal{D})$ with $m = n$. This search algorithm can be turned into the following decision algorithm: If the assignment a produced by the algortihm is satisfiable for the input formula ϕ, output "yes". If not, say "I don't know". The advantage of this algorithm over the trivial one that always outputs "yes" is that it is errorless: it never produces an incorrect answer. Every once in a while it says it doesn't know the answer, but this happens with very small probability over the choice of the formula ϕ.

Definition 3. The decision heuristic is *errorless* if for all x such that $A(x) \neq f(x)$, $A(x)$ outputs "I don't know".

Errorless heuristics are also interesting for distributional NP problems that almost never have a solution (i.e., $f(x)$ is zero for most x). In random 3SAT with $m = 10n$, an overwhelming fraction of the instances are unsatisfiable so there is a trivial decision heuristic, but no errorless heuristic is known.

The definitions we gave are for deterministic algorithms. They can be extended to randomized algorithms by taking the probability also over the algorithm's random tape. For errorless heuristics there is the additional requirement that the wrong answer is never the most probable one: For every input x, $A(x)$ outputs $f(x)$ or "I don't know" with probability at least $3/4$.

It remains to say a word about which distributions we will consider as legitimate in the context of efficient average-case computation. We will say that a distributional ensemble $\mathcal{D}$ is *polynomial-time samplable* if there is a randomized polynomial-time Turing Machine that when given any string of length n as its input outputs a sample of D_n. The class *distributional search NP* consists of all pairs $(S, \mathcal{D})$ where S is an NP-search problem and $\mathcal{D}$ is a polynomial-time samplable ensemble. We similarly define distributional decision NP.

2 Hardness amplification

In worst-case complexity the goodness of an algorithm is measured by a single quantity, its running time. To talk about average-case performance we had to introduce another parameter: the failure probability $\varepsilon(n)$. This makes it more difficult to compare two algorithms as one may have a superior running time but higher failure probability.

Even though for specific problems the choice of failure probability can make a difference, it turns out that for the class of NP as a whole it does not matter much: If all distributional NP problems can be efficiently solved even on a small fraction of inputs, then they can be solved on almost all

[2] This argument is technically flawed as the probability could conceivably jump from almost 1 to almost 0 when a single random clause is added, but it works for a related distribution with a continuous parameter. Experiments indicate that this m is around $4.26n$.

inputs. We illustrate this phenomenon in the context of search problems. Similar results hold for decision problems, with respect to both errorless and general heuristics, but their proofs are more involved.

To explain how this works it helps to think of the contrapositive of the above statement: If there exists a distributional NP search problem $(S, \mathcal{D})$ that is hard to solve by efficient algorithms on even a small fraction of inputs, then there exists another problem $(S', \mathcal{D}')$ that is hard on almost all inputs. This is where the name "hardness amplification" comes from: We take a mildly hard problem S and create a much harder one S'. The main idea is the following: If one instance of S is a bit hard to solve, then many independent instances should be very hard to solve. This suggests defining S' as consisting of many instances of S:

$$((x_1, \ldots, x_k), (y_1, \ldots, y_k)) \in S' \quad \text{if and only if} \quad (x_i, y_i) \in S \text{ for all } 1 \leq i \leq k.$$

for a sufficiently large k depending on n (to be specified later). The distribution D'_n for size n instances in S' consists of k independent copies of D_n. Clearly S' is an NP-search problem if S is one and $\mathcal{D}'$ is polynomial-time samplable if $\mathcal{D}$ is and k is some polynomial of n.

Now we want to argue that if S is hard, then S' is much harder. For contradiction suppose there is an efficient algorithm A' that solves S' on at least an ε-fraction of inputs from D'_n, namely

$$\Pr_{x_1 \sim D_1, \ldots, x_k \sim D_k}[((x_1, \ldots, x_n), (y_1, \ldots, y_n)) \in S'] \geq \varepsilon$$

where $(y_1, \ldots, y_n)$ is the output of $A(x_1, \ldots, x_n)$ (parsed in the correct form).

Given an instance x for S we now want to use A' to solve it. We apply the following algorithm:

A: On input x,
Sample $x_1, \ldots, x_n$ independently from D_n.
For all i at random between 1 and k:
Compute $(y_1, \ldots, y_{i-1}, y, y_{i+1}, \ldots, y_k) = A(x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_k)$.
If $(x, y) \in S$, output y.

Let's say an input x is *good* if $A(x)$ is a solution for x in S with probability at least δ over the randomness of A. As long as δ is not negligibly small, its exact value is not terribly relevant as we can increase this probability by trying out $A(x)$ many times independently. Then the chances of it finding a solution increase dramatically: If we perform $O((1/\delta)\log(1/\gamma))$ repetitions, the probability goes up from δ to $1 - \gamma$.

Now let $\mathbf{x} = (x_1, \ldots, x_k)$ be a sequence of k independent samples from D_n. For A' to solve $\mathbf{x}$ one of two things must happen: Either all x_i must be good, or A' solves $\mathbf{x}$ but x_i is not good for some i. By a union bound,

$$\begin{aligned}\Pr_{\mathbf{x} \sim D'_n}[A' \text{ solves } \mathbf{x}] &\leq \Pr[\text{all } x_i \text{ are good}] + \sum_{i=1}^{k} \Pr[A' \text{ solves } \mathbf{x} \text{ and } x_i \text{ is not good}] \\ &\leq \Pr[\text{all } x_i \text{ are good}] + \sum_{i=1}^{k} \Pr[A' \text{ solves } \mathbf{x} \mid x_i \text{ is not good}].\end{aligned}$$

Since the x_i are independent samples from D_n, the first probability is just $\Pr_{x \sim D_n}[x \text{ is good}]^k$. We bound the second probability like this: If x_i is not good then A' must have failed to find a solution with probability δ in the i-th run of A on input x_i. But this input to A' was sampled exactly from the product distribution D'_n conditioned on x_i, so

$$\Pr[A' \text{ solves } \mathbf{x} \mid x_i \text{ is not good}] < \delta.$$

By our definition of good, the second probability is less than δ. After reversing the inequality we obtain

$$\Pr_{x\sim D_n}[x \text{ is good}] \geq (\varepsilon - k\delta)^{1/k}$$

so if we choose $\delta = \varepsilon/2k$ we obtain that x is good with probability at least $(\varepsilon/2)^{1/k}$, which is at least $1 - (1/k) \cdot \ln(2/\varepsilon)$.

In conclusion, for any parameters γ and ε we can choose values of k and $1/\delta$ that grow polynomially in $1/\gamma$ and $1/\varepsilon$ so that if A' solves $(S', \mathcal{D}')$ on an ε-fraction of inputs then A solves $(S, \mathcal{D})$ on a $(1-\gamma)$-fraction of inputs. Moreover, the running time of A' is polynomial in these parameters and the running time of A. Therefore

Theorem 4. *For every distributional NP-search problem $(S, \mathcal{D})$ and all polynomials p and q there exists a distributional NP-search problem $(S', \mathcal{D}')$ such that if S' has a randomized polynomial-time heuristic with error at most $1-1/p(n)$ on inputs from D'_n then S has a randomized polynomial-time heuristic with error at most $1/q(n)$ on inputs from D_n.*

3 Decision versus search for distributional NP

The search-to-decision reduction for NP from the last lecture does not work in the distributional setting: Whenever an input bit to the circuit is fixed, the distribution on circuit changes. We now describe a different reduction that also works for distributional problems and proves the following:

Theorem 5. *For every distributional NP search problem $(S, \mathcal{D})$ there is a distributional NP decision problem $(f, \mathcal{D}')$ such that if $(f, \mathcal{D}')$ has a randomized polynomial-time decision heuristic with error at most $\varepsilon(n)$ then $(S, \mathcal{D})$ has a randomized polynomial-time search heuristic with error at most $O(p(n)\varepsilon(n))$ for some polynomial p.*

Let us first think of the special case where every instance x of S has at most one solution y. Then we can extract this solution by asking the questions "What is the j-th bit of y?" for all indices j. We let f be the decision problem

$$f(x,j) = \begin{cases} y_j, & \text{if there exists a } y \text{ such that } (x,y) \in S \\ 0, & \text{if not} \end{cases}$$

which is in NP. To solve S on input x, query the algorithm A for f on inputs $(x, 1)$ up to (x, m) and concatenate the answers. Here m is some upper bound on the length of solutions; if x is sampled from D_n, then m grows at most polynomially in n.

Clearly if A solves all instances of f then we can find all unique solutions to S. Now suppose A fails on at most an ε-fraction of inputs (x, j) where x is sampled from D_n and j is uniformly distributed in the range $\{1, \ldots, m\}$. Then

$$\begin{aligned} \Pr_{x\sim D_n}[A(x,j) \neq f(x,j) \text{ for some } j] &\leq \sum_{j=1}^{m} \Pr_{x\sim D_n}[A(x,j) \neq f(x,j)] \\ &= m \cdot \Pr_{x\sim D_n, j\sim\{1,\ldots,m\}}[A(x,j) \neq f(x,j)] \\ &\leq m\varepsilon \end{aligned}$$

so solutions to S can be found in polynomial time with error $m\varepsilon$.

If x has more than one solution this reduction doesn't guarantee anything because the output of the reduction may consists of parts of different solutions. In the general case, we we will impose additional constraints on S that guarantee solutions are unique at least some of the time.

Suppose that some instance x of S has K solutions; K can be any number between 0 and 2^m and this number may depend on x. If we knew this K and sampled every possible $y \in \{0,1\}^m$ with probability $1/K$ independently at random, then the probability that among the ys that were sampled only a single solution to x survives is at least some constant greater than zero (it approaches $1/e$ as K grows). If we can arrange that only those y that survive the sampling are counted as solutions then some constant fraction of the inputs have unique solutions and these can be recovered by solving the decision problem f. There are two issues to resolve in order to turn this idea into a reduction.

First, the value of K is not known in advance. You can convince yourself that the reasoning still works if the value of K was known only approximately, say it was estimated by the power of two that is closest to K. This approximation may still not be known, but now there are only m and not 2^m possibilities to try.

Second, it is not clear how to formulate the constraint "$(x, y) \in S$ and y survives the sampling" as an NP-relation. The sampling has "exponential complexity" as we need to specify the survival of every potential solution y from $\{0,1\}^m$. This can be greatly reduced by correlating the outcomes of different ys. To describe the proper way to do this we take a detour to discuss pairwise independence.

Pairwise independence A collection of random variables $H_1, \ldots, H_M$ is *pairwise independent* if H_i and H_j are independent when $i \neq j$. Pairwise independence is a weaker property than independence. However, specifying M fully independent random bits requires M bits of information, while we shall see that for pairwise independence $2\lceil \log N \rceil$ bits of information are sufficient.

Here is an (important) example when $M = 2^m$ for some m. Then there is one random variable for every string of length m, so we can think of H as a randomized function whose input comes from $\{0,1\}^m$. The function H is determined by two uniformly random and independent elements a and b of the finite field $\mathbb{F}_{2^m}$ and its value is $H(y) = ay + b$ (where the arithmetic is over $\mathbb{F}_{2^m}$). First, for every y, $H(y)$ is uniformly distributed over all 2^m possible values because even when a is fixed, the random shift b ensures uniformity. Second, $H(y)$ and $H(y')$ are independent whenever $y \neq y'$: By linearity, $H(y)$ and $H(y')$ uniquely specify a and b, so the probability of any outcome $(H(y), H(y'))$ is 2^{-2m} which is the product of the probabilities of observing $H(y)$ and $H(y')$ separately. In particular, H viewed as a randomized function from $\{0,1\}^m$ to $\{0,1\}^m$ satisfies the following definition:

Definition 6. A randomized function $H\colon Y \to Z$ is a *pairwise independent hash function* if $H(y)$ is uniform for all y and $H(y), H(y')$ are independent for all $y \neq y'$.

Since both uniformity and independence are preserved by projection, functions obtained by dropping some input or output bits of H are also pairwise independent. We can now show that pairwise independence is sufficient for isolating unique elements of large sets.

Lemma 7. *Let $H\colon \{0,1\}^m \to \{0,1\}^k$ be a pairwise independent hash function. For every subset Y of $\{0,1\}^m$ of size between 2^{k-2} and 2^{k-1}, the probability that there is a unique y in Y such that $H(y) = 0$ is at least $1/8$ (over the choice of H).*

Proof. The event "there is a unique $y \in Y$ such that $H(y) = 0$" is a *disjoint* union of the $|Y|$ events

"$H(y) = 0$ and $H(y') \neq 0$ for every $y' \in Y - \{y\}$", one for each y in Y. Therefore

$$\begin{aligned}
&\Pr[\text{there is a unique } y \in Y \text{ with } H(y) = 0] \\
&\quad = \sum_{y \in Y} \Pr[H(y) = 0 \text{ and } H(y') \neq 0 \text{ for all } y' \in Y - \{y\}] \\
&\quad = \sum_{y \in Y} \Pr[H(y') \neq 0 \text{ for all } y' \in Y - \{y\} \mid H(y) = 0] \Pr[H(y) = 0] \\
&\quad = \sum_{y \in Y} (1 - \Pr[H(y') = 0 \text{ for some } y' \in Y - \{y\} \mid H(y) = 0]) \Pr[H(y) = 0] \\
&\quad \geq \sum_{y \in Y} \left(1 - \sum\nolimits_{y' \in Y - \{y\}} \Pr[H(y') = 0 \mid H(y) = 0]\right) \Pr[H(y) = 0] \\
&\quad \geq |Y| \cdot (1 - |Y| \cdot 2^{-k}) \cdot 2^{-k}.
\end{aligned}$$

Since Y has size between 2^{k-2} and 2^{k-1} the last expression is at least $(1/4) \cdot (1 - 1/2) = 1/8$. □

Search-to-decision reduction We can now prove Theorem 5. Given $(S, \mathcal{D})$, we define

$$f(x, j, h, k) = \begin{cases} y_j, & \text{if there exists a } y \text{ such that } (x, y) \in S \text{ and the first } k \text{ bits of } h(y) \text{ are zero} \\ 0, & \text{if not} \end{cases}$$

As before, j is an index ranging from 1 to m. Now h is the description of a pairwise indepependent hash function from $\{0,1\}^m$ to $\{0,1\}^{m+2}$ (given by the elements a and b in the above construction) and k is a number between 2 and $m + 2$. A sample (x, j, h, k) from D'_n is obtained by sampling x from D_n and choosing j, h, and k uniformly at random from their respective sets.

Now assume A solves $(f, \mathcal{D}')$ with error ε. The algorithm for S chooses h at random and runs A for all choices of j and k. If any choice of k gives a solution y to S, it outputs this solution.

By the same analysis as above,

$$\Pr_{x,h}[A(x, j, h, k) \neq f(x, j, h, k) \text{ for some } j, k] \leq m^2 \varepsilon.$$

We can now say that with probability at most $16m^2\varepsilon$ over the choice of x,

$$\Pr_h[A(x, j, h, k) \neq f(x, j, h, k) \text{ for some } j, k] > \frac{1}{16} \tag{1}$$

for otherwise the combined probability over x and h would exceed $m^2\varepsilon$. Now assume that the complement holds for x, namely the probability in (1) is at most 1/16. By Lemma 7, the probability that there is a unique y such that (x, y) is in S and the first k bits of $h(y)$ are zero when $k = \lfloor \log(\text{number of witnesses } y \text{ for } x) \rfloor - 1$ is at least 1/8. So the probability that there is a unique witness y *and* A is correct on all j and k is at least 1/16 over the choice of h. If this happens then $(A(x, 1, h, k), \ldots, A(x, m, h, k))$ must equal this y and the algorithm for S succeeds.

To summarize, we showed that with probability $1 - 16m^2\varepsilon$ over the choice of instance x from D_n the algorithm for S finds a solution with probability at least 1/16 over the randomness of the algorithm. This proves Theorem 5.

4 Completeness for distributional problems

Our proof of the completeness of 3SAT from the last lecture can be interpreted as a reduction between distributional problems. In the language of average-case complexity it says that for every

distributional NP problem $(S, \mathcal{D})$ there is a samplable ensemble $\mathcal{D}'$ such that if $(3\text{SAT}, \mathcal{D}')$ has a polynomial-time heuristic then $(S, \mathcal{D})$ also has one with the same error. Instances from D'_n are obtained by sampling an instance from D_n and running the reduction on it. Therefore if we can solve 3SAT heuristically on all efficiently samplable ensembles then we can solve any problem in distributional NP. It is a bit unsatisfying, however, that we need a different algorithm for every distribution. Is there was a single, "hardest" distribution for 3SAT instances?

Let's ask an easier question first. We just argued that every distributional NP problem can be reduced to 3SAT under some ensemble depending on it. Can we reverse the role of the problem and the ensemble? Namely, is there a hard ensemble $\mathcal{H}$ such that every distributional NP problem reduces to some other NP problem under the distribution $\mathcal{H}$?

Let's take the random 3CNF ensemble $\mathcal{D}$ from Section 1 as an example. Recall that a sample from D_n is obtained by choosing m uniformly random clauses out of the $8\binom{n}{3}$ possibilities on n variables. We can represent such an instance as a string of m blocks, each consisting of a $\log 8\binom{n}{3}$-bit string specifying the choice of clause. Then a uniform choice of this string represents exactly a sample from the distribution D_n.[3] So the exact choice of distribution in this example is a question of representation and the uniform distribution works as well as any other.

While this example is misleadingly simple it turns out that it is always possible to "represent" inputs by uniformly random strings if we modify the problem:

Theorem 8. *For every distributional NP search problem $(S, \mathcal{D})$ there exists a search problem S' and a polynomial p such that if $(S', \mathcal{U})$ has a randomized polynomial-time heuristic with error at most $\varepsilon(n)$ then $(S, \mathcal{D})$ has a randomized polynomial-time heuristic with error at most $p(n)\varepsilon(n)$.*

Here $\mathcal{U}$ is the ensemble of uniform distributions on all input lengths in order. In your next homework you will show that there exists a problem BH that is complete for the uniform ensemble: Every $(S, \mathcal{U})$ reduces to $(\text{BH}, \mathcal{U})$ (up to a polynomial loss in error). So this problem is "complete" for distributional NP.

If all NP problems have efficient worst-case algorithms then clearly all distributional NP problems have efficient heuristics. It is not known if the converse is true.

5 One-way functions

A one-way function is a function that is easy to compute, but hard to invert.

Definition 9. A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is (t, ε)-one-way if f is polynomial-time computable but the distributional search problem "find x such that $f(x) = y$", where y is obtained by applying f to a uniform sample in $\{0,1\}^n$ cannot be solved by any heuristic that runs in time $t(n)$ with error at most $1 - \varepsilon(n)$ on inputs of size n.

We usually think of t and $1/\varepsilon$ as functions that grow faster than any polynomial, so the task of inverting f is much harder than the task of computing f. Alice can easily generate a random $r \sim \{0,1\}^n$ and send the value $f(r)$ to Bob, but Bob cannot efficiently find any x such that $f(x) = f(r)$ with noticeable probability. (It is unfair to require Bob to recover r itself, which may be underspecified by the value $f(r)$, so we allow him to come up with any x that has the same image as r.)

[3] We have unrealistically assumed here that $8\binom{n}{3}$ is an exact power of two. The best we can hope for is to make the two distribution "close" up to some small error, but we will disregard this issue.

Inverting a one-way function is a special type of problem in distributional NP. For this reason, one-way functions are not known to exist: If any particular f was proved to be one-way, then the task of inverting f would be a provably hard NP problem, certifying that P does not equal NP.

However, there are many examples of functions that are believed to be one-way. Here is one example. Let x be a uniformly random assignment in $\{0,1\}^n$ and $\phi_1, \ldots, \phi_n$ be the following random formulas: Each ϕ_i is of the form $MAJORITY(x_a, x_b, x_c)$ with the indices a, b, c chosen uniformly and independently within and among the clauses. Then the function

$$f(x, \phi_1, \ldots, \phi_n) = (x, \phi_1(x), \ldots, \phi_n(x))$$

is easy to compute, but the best known heuristics for inverting it require time exponential in n.

In cryptography it is sometimes required that inverting f be hard not for algorithms running in time $t(n)$, but for circuit families of size $s(n)$. Since algorithms can be simulated by circuits with polynomial loss, but not the other way around, inversion hardness against circuits is potentially more difficult to satisfy. However I do not know of any example that distinguishes between the two definitions. In fact I do not know of any example of a problem in NP that is believed to be easy for polynomial-time circuits but hard for polynomial-time algorithms in the worst case or in the distributional setting.

References

The presentation of this lecture is based on the survey *Average-case complexity* by myself and Luca Trevisan. The first definitions were given by Levin. The ones here are somewhat different and more in line with modern practice. Theorem 4 was stated (but not proved) by Yao. Theorems 5 is by Ben-David, Chor, Goldreich and Levin. Theorem 8 is by Impagliazzo and Levin. The example in Section 5 is a special case from a class of functions all conjectured to be one-way by Goldreich.