CSE696 Week 4: Counting Classes and Probabilistic Computation

The question "how many solutions are there?" subsumes the question "is there a solution?" The surprise is that it also captures all of the alternation in the polynomial hierarchy, as shown by the theorem of Seinosuke Toda in 1988-89 that PH is contained in BP[$\oplus$P], which in turn is contained in P$^{\#\text{P}}$. Well, this tells us we have lots more things to define, so let's get to it---first in sections **9.1--9.2** of Arora-Barak, then **Chapter 7**, then Toda's theorem in section **9.3**.

Definition 1: A function $h : \Sigma^* \rightarrow \mathbb{N}$ belongs to #P (pronounced "sharp-P" more often than "number-P") if there is a predicate $R(x,y)$ in P and a polynomial p such that for all x,
$h(x) = ||\{y : |y| \leq p(|x|) \wedge R(x,y)\}||$.

We can abbreviate this by writing $h(x) = \#^p y. R(x,y)$ to mimic the "lambda style" $\lambda z. A(z)$ style of writing functions. (But, maybe we will not need to do so.) We have already defined the notion of $g \leq_m^p h$ as there being a function $f \in$ FP such that $g = h \circ f$. [Not as $f \circ h$, curiously.] That is, $g(x)$ equals the number of solutions to the predicate $R_h(x', y)$ on $x' = f(x)$. Compare: For languages A, B viewed as 0-1 valued functions, $A \leq_m^p B$ via f means $A(x) = B(f(x))$.

Theorem 1: The function ***#sat***(ϕ) = the number of satisfying assignments of the unquantified propositional Boolean formula ϕ is complete for #P under $\leq_m^p$.

Proof: That $\#sat$ belongs to #P is immediate from the predicate "a satisfies ϕ" being linear(?) time decidable. Let any $h \in$ #P defined by $R(x,y)$ and $p(n)$ be given. The essence is that the Cook-Levin construction of $C_n(x,y)$ and then the formula ϕ_n is **parsimonious**, meaning that once x is fixed, the number of y such that $R(x,y)$ holds equals the number of staisfying assignments to the resulting formula ϕ_x. The reason is that everything in the body of ϕ_x is equational, so that any assignment to $y_1, \dots, y_p$ forces the values of all other variables, up through the output wire variable w_o. (This also holds true under relativizations, so for any oracle A, $\#sat^A$ is complete for #P^A under many-one reductions in FP---without needing to use FP^A.) ☒

It is natural first to ask, what *language* class is commensurate with #P? We've already mentioned P$^{\#\text{P}}$, which equals P$^{\#sat}$. We will see some senses by which this may be bigger than necessary, just like P$^{\text{NP}}$ is bigger (or so we believe) than NP. Hence our second question is, what about languages that polynomial-time *many-one* reduce to $\#sat$. This leads to a type-compatibility issue insofar as g has range $\{0,1\}$ but h in $g = h \circ f$ has range $\mathbb{N}$, and this will soon lead us to consider the subclass of NP called UP. The issue is "finessed" by starting with langauge(s) that are equivalent to evaluating $h(x)$. Here are three choices:

1. $G_h = \{(x,r) : r = h(x)\}$. This is standardly called the "graph" of the function h.
2. $L_h = \{(x,r) : r \leq h(x)\}$. This is the "less-than-or-equal graph."
3. $P_h = \{(x,u) :$ u is a prefix of the function value $h(x)$ in standard binary notation$\}$.

Using the graph G_h may seem most natural. But can we compute $h(x)$ in polynomial time using G_h as oracle? Using G_h leads to a complexity class called $C_=P$, which is not well understood even now (IMHO).

The other two languages do work as oracles for computing $h(x)$ numerically in polynomial time (quadratic time as far as the oracle usage is concerned, per the footnote toward the end of the Week 3 notes). With L_h one uses binary search, which is the more numerically natural process. Now we can give a simpler equivalent definition to the standard one:

Definition 2: A language L belongs to PP if and only if $L \leq_m^p L_h$ for some function h in #P, via a polynomial-time computable function $f(x) = (x', r)$ in which $|x'|$ depends only on $|x|$.

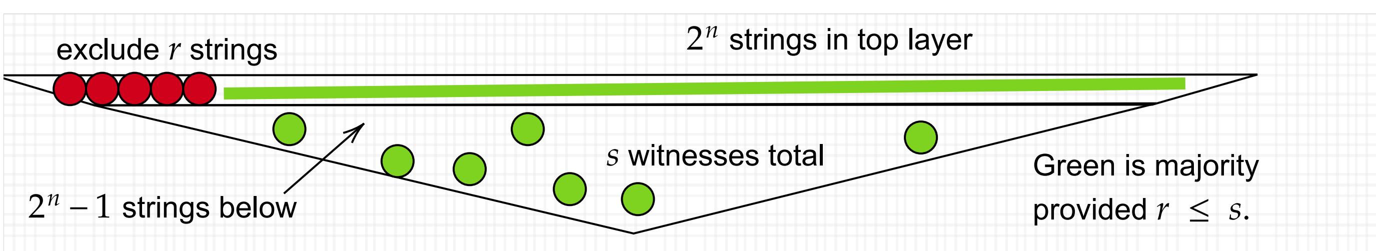
The clause with "via" is technical but makes it easier to prove a lemma en-route to the standard definition and its interpretation.

Lemma 2: For any $L \in$ PP we can find a polynomial-time predicate $R(x, y)$ and a polynomial p such that for all x, $x \in L$ if and only if a majority of strings y of length up to $p(n)$ give $R(x, y)$.

Proof: By h in #P, we can take $R_0(x, y)$ and p_o defining h in #P such that $L \leq_m^p L_h$ via a polynomial-time computable function f. Let us use the notation $(x', r) = f(x)$ and $n' = |x'|$ depending only on $n = |x|$. Take $p(n) = p_0(n') + 1$. A useful point to note is that the number of binary strings y of length up to $p(n)$ is odd, and that those of length exactly $p(n)$ make up a bare majority. So we need only arrange that if there are at least r witnesses of the original predicate $R_0(x', y)$ with $|y| \leq p_0(n') = p(n) - 1$, then those together with $2^{p(n)} - r$ witnesses of length $p(n)$ make up a majority. Accordingly, define

$$R(x, y) = \mathit{if}\ |y| < p(|x|)\ \mathit{then}\ R_0(x', y)\ \mathit{else}\ y \geq w_r,$$

where w_r means the string in $\{0,1\}^{p(n)}$ with binary expansion r. To see why this works, let s be the actual value of $h(x')$, that is, the number of y giving $R_0(x', y)$. If $r = s$, which is the greatest r for which $(x', r) \in L_h$, then we get s witnesses y of length $< p(n)$ from $R_0(x', y)$ and exactly $2^{p(n)} - r = 2^{p(n)} - s$ witnesses y of length $p(n)$, adding up to exactly $2^{p(n)}$. This is a bare majority, as required to say $x \in L$. If $r < s$, then (x', r) is still in L_h, so we want to accept x, which happens because we get even more witnesses of length $p(n)$ from the lower value of r. But if $r > s$ then the number of witnesses comes short of a majority. ☒



[Self-study exercise: Work this out without the convenience of n' depending only on n.]

[**Saturday's lecture will pick up by reviewing the lemma just above with my new picture, then continue as below---maybe getting to the matrix example but short of defining** BPP.]

The same idea applied with $r = 1$ and the "majority" formulation yield the following.

Proposition 3: NP $\subseteq$ PP. ☒

Proposition 4: PP is closed under complements, so also co-NP is contained in PP. ☒

For a long time, it was unknown whether PP is closed under intersection. This was originally shown via algebra. Then Scott Aaronson used the algebra to give a more conceptual explanation by showing that PP equals "BQP with postselection"---a class which is just as obviously closed under all Boolean operations as P^{PP} is. [We'll appreciate this and how the "$1/2 + o(1)$" bare-majority advantage of PP gets related to "$1 - o(1)$" advantage of the more-powerful quantum post-selection operation after we cover BPP. The unfair power of post-selection comes from being able to condition $\Pr[y = 1 | w = 1]$ on events w whose probability of being 1 is positive but exponentially small.] In consequence, the Boolean closure of NP is contained within PP. Yet we still do not know whether $\mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{PP}$, let alone higher parts of the polynomial hierarchy, in contrast to the theorem that $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$, which we will cover upon jumping back to chapter 9 in section 9.3.

Note also: $\mathsf{P}^{\#\mathsf{P}} = \mathsf{P}^{\mathsf{PP}}$ because we can execute the binary search procedure via extra oracle calls. But they may not equal PP by itself. The relationship between PP and P^{PP} is like that between many-one reductions and Turing reductions quite in general, as will come out next. We do have $\#\mathsf{P} = \mathsf{FP} \iff \#sat \in \mathsf{FP} \iff \mathsf{P}^{\mathsf{PP}} = \mathsf{P} \iff \mathsf{PP} = \mathsf{P}$.

Counting and Predicates

Here is a state of affairs that has caused considerable confusion: A language $L \in$ NP can come with many different witness predicates $R(x, y)$, which we tacitly suppose to be bundled with length-bounding polynomials p. Do we consider L alone or (L, R) to be "the thing"? Well, R should be *the* thing because it uniquely induces L as $L_R = \left\{x : \left(\exists^p y\right) R(x, y)\right\}$. Once R is specified, we get the counting function $\boldsymbol{h_R(x)} = \#^p y. R(x, y)$, but note that it pertains to R, not alone to L.

1. Every (L, R) gives a parsimonious reduction from h_R to $\#sat$.
2. If L is NP-complete via an invertible reduction g from SAT, then L has a witness predicate R' such that the induced reduction from $\#sat$ to $f_{R'}$ is parsimonious. Namely: $R'(x, y) = sat\left(g^{-1}(x), y\right)\ if\ x \in ran(g)\ else\ R(x, y)$.

3. There are invertible NP-complete languages L whose "natural" witness predicate R definitely does not allow a parsimonious reduction from $\#sat$ to h_R. For example, whether a graph is 4-edge colorable is NP-complete, but the only graph having a unique 4-edge coloring (not counting permutations of the colors) is the star with 5 nodes and four points.
4. Whether there are NP-complete languages without invertible reductions from SAT is a subversive question. The **Berman-Hartmanis Conjecture** asserts that all NP-complete languages are **p-isomorphic**, meaning equivalent under polynomial-time computable and invertible permutations of Σ^*.
5. If $\#sat$ many-one reduces to h_R, meaning there is a function $f \in FP$ such that for all Boolean formulas ϕ, $\#sat(\phi) = h_R(f(\phi))$, then we have:

$$\phi \in SAT \iff \#sat(\phi) \geq 1 \iff h_R(f(\phi)) \geq 1 \iff \#^p y.\, R(f(\phi), y) \geq 1 \iff f(\phi) \in L_R \,,$$

   so L_R is NP-complete.
6. However, there are polynomial-time predicates R such that f_R is #P-complete under poly-time ***Turing*** reductions and yet L_R belongs to P. The most amazing one IMHO is counting the number of satisfying assignments to a 2CNF formula ψ *with no negated variables*. Such a ψ is trivially satisfiable, let alone that 2SAT belongs to P. The most historically important such problem is the following one.

The "N-Rooks Problem": Given an $N \times N$ chessboard in which every square is marked either 0 or 1, can we place N rooks on the squares marked 1 so that no two rooks attack each other?

This is both easier and harder than the famous N-Queens Problem: the latter allows you to use every square of the chessboard but queens can attack each other diagonally too. Well, chess is a red herring here---the problem has two more familiar interpretations.

Bipartite Perfect Matching: Given an $N \times N$ bipartite graph (V_1, V_2, E), can we find N edges that connect every vertex in V_1 to a distinct node in V_2?

Binary Permanent: Given an $N \times N$ binary matrix A, can we find a nonzero *diagonal product*, so that $perm(A) > 0$?

The permanent function is what you get from the formula for the determinant if you "simplify" it by removing the minus signs. That is, letting S_N denote the set of permutations of N elements:

$$\det(A) = \sum_{\sigma \in S_N} \prod_{i=1}^{N} (-1)^{sign(\sigma)} A[i, \sigma(i)]$$

$$perm(A) = \sum_{\sigma \in S_N} \prod_{i=1}^{N} A[i, \sigma(i)]$$

Now despite the fact that the determinant is computable in polynomial time (indeed, the same order of time it takes to multiply two $N \times N$ matrices), the "simpler" function $perm(A)$ is NP-hard. Unlike "NP-complete", the term "NP-hard" usually refers to polynomial-time Turing reductions. The term "#P-complete" always requires specifying the reductions because the Turing case has so much influence---indeed, Arora-Barak define it that way in section 9.2. This arguably stems from the famous theorem that explained why trying to find an easy procedure for computing $perm(A)$ had met with a century of failure.

Theorem 5 [Leslie Valiant, late 1970s]: The permanent function (of 0-1 matrices or more generally) is complete for #P under polynomial-time Turing reductions.

I will skip the proof given later in chapter 9 by Arora and Barak---we will do #P-completeness under $\leq_m^p$ by extending $\#sat$ into algebraic functions (related to quantum circuits) instead. But it contrasts with the famous theorem that did much to coalesce the feeling about P as being the benchmark class for "feasibility" to begin with:

Theorem 6 [Edmonds, 1965; before?*]: **[Bipartite] Perfect Matching** is in P.

[*What Edmonds actually did was prove that for every non-maximal matching in a bipartite graph there is a path that begins and ends with unmatched nodes and alternates edges in and not in the matching. Flipping the in/not-in status of the edges in that path then yields a bigger matching. Finding such a path in polynomial-time, as Edmonds showed how to do, then yields a poly-time algorithm for the whole problem. Then earlier algorithms were later proved to operate in polynomial time as well.] It remains a philosophical mystery why finding a perfect matching (or telling that one doesn't exist) is easy but counting them is hard. We, however, move on to this:

Equation Solving: Given polynomial equations $p_1(x_1, \dots, x_n) = 0, \dots, p_s(x_1, \dots, x_n) = 0$ over a field $\mathbb{F}$, is there a common solution (and if so, how many)?

The challenge is not so much to prove this #P-complete (usually under $\leq_m^p$) as to find cases that are *not* #P-complete. If the equations include $x_1^2 - x_1 = 0$ through $x_n^2 - x_n = 0$, then we are down to asking about solutions in which each x_i is restricted to be 0 or 1. Call this the "binary restriction". Then just replace each 3CNF clause in an instance of 3SAT by a corresponding degree-3 equation---for instance, $(x_i \vee \bar{x}_j \vee x_k)$ becomes $(1-x_i)x_j(1-x_k) = 0$.

Under the binary restriction, one can even reduce from $\#sat$ to equations that are *linear*. This does not contradict the polyniomial-time solvability of general linear equations because the equations defining the restriction are quadratic. When the 0-1 property applies also to the possible *values*, one can also multiply all the equations together in the form

$$(1 - p_1) \cdots (1 - p_s) - 1 = 0,$$

thus getting a single multi-variable polynomial equation---albeit a polynomial of degree on the order of s

that would have exponentially many terms if you multiplied it out. There are relaxations of the 0-1 property on values and/or arguments that also make this work. They all have parsimonious reductions from $\#sat$.

One more note before moving on: There are numerous other restrictions one can place on equation-solving problems. A famous case where the counting problem does belong to P is counting solutions to
a single quadratic polynomial over the binary field $\mathbb{F}_2$. Make it a quadratic polynomial with values modulo 4, however, and the solution-counting problem "sproings back" to being #P-complete (under many-one reductions). Following on from newer work by Valiant, Jin-Yi Cai has made a large-scale project of studying this easy-or-complete "Dichotomy" phenomenon. It is IMHO even more compelling than the paucity of "natural" problems that are believed to be neither in P nor NP-complete, of which Factoring, Graph Isomorphism, and the Minimum Circuit Size Problem (given a binary string z of length $n = 2^k$, and a number r, in there a k-input circuit C of size at most r such that for all $i \leq n$, z_i equals the value of C on the i-th element of $\{0,1\}^k$?) are the only ones with staying power, IMHO. But we will be in a position to ask whether quantum circuits may furnish a broad intermediate class of algebraic problems between P and #P-complete.

Bounded-Error Probabilistic Computation

Let's first motivate this with an example I used also in CSE596---it also connects to the note just above:

For any natural number m, $\mathbb{Z}_m$ stands for the integers modulo m. If m is a prime number p, then $\mathbb{Z}_p$ is a *field* (so that one can divide as well as multiply) and we write it as $\mathbb{F}_p$. The simplest such case is $p = 2$ which is $\{0,1\}$ with the usual addition modulo 2 and multiplication. The field structure helps us prove the following result more easily.

Lemma 7: Suppose A, B, C are $n \times n$ matrices over $\mathbb{F}_p$ such that $AB \neq C$. Then

$$\Pr_{u \in \mathbb{F}_p^n}[ABu \neq Cu] \geq \frac{p-1}{p}.$$

Proof: Write $D = AB - C$. Note that we are not going to *calculate* D, because that would take the (standardly cubic) time for multiplying A and B that we are trying to avoid, but we are allowed to *argue based on its existence*. By linearity, $ABu \neq Cu \iff Du \neq 0$. So D has at least one row i with a nonzero entry, and its use may give a nonzero entry in the i-th place of the column vector $v = Du$. Note that

$$v_i = \sum_{j=1}^{n} D[i,j]u_j .$$

Let j_0 be a column in which row i has entry $c = D[i, j_0] \neq 0$. For any vector u, we can write

$$v_i = cu_{j_0} + a \quad where \quad a = \sum_{j \neq j_0} D[i,j]u_j.$$

The key observation is that because $\mathbb{F}_p$ is a field, for any $c \neq 0$, the values cu_{j_0} run through all p possible values as u_{j_0} runs through all p possibilities. Regardless of the value of a determined by the rest of row i and the rest of the vector u, the values $cu_{j_0} + a$ run through all p possibilities with equal probability. Hence the probability that $v_i \neq 0$ is exactly $\frac{p-1}{p}$. The probability of getting $v \neq 0$ (which could come from other nonzero entries too) is at least as great. ☒

The upshot is:

- If $AB = C$ then you will never be deceived: you will always get equal values from $A(Bu)$ and Cu and will correctly answer "yes, equal."
- If $AB \neq C$ and you try k vectors u at random, if you ever get $A(Bu) \neq Cu$ then you will know to answer "no, unequal" with 100% confidence.
- If you get equality each time, you will answer "yes, equal" but there is a $\frac{1}{p^k}$ chance of being wrong.

If you consider, say, a 1-in-n^3 chance of being wrong as minuscule, then you only need to pick k so that $p^k > n^3$, so $k = \frac{3}{\log p} \log n$ will suffice. Presuming p is fixed, this means $O(\log n)$ trials will suffice. The resulting $O\left(n^2 \log n\right) = \widetilde{O}\left(n^2\right)$ running time handily beats the time for multiplying AB out. Thus we trade off *sureness* for *time*.

For arithmetic modulo m not prime, or without any modulus, the analysis is messier---but not only is the essence the same, but the asymptotic order of k in terms of n and the confidence target $\epsilon(n)$ is much the same---it didn't really depend on p to begin with.