(1) For any a, $0 < a < 1$, define PP_a to be the class of languages L such that for some polynomial $p(n)$ and predicate $R(x,y)$ decidable in time $p(|x|)$, and all x,

$$x \in L \iff \Pr_{|y|=p(|x|)}[R(x,y)] > a.$$

Show that $\mathrm{PP}_a = \mathrm{PP}$. Does this hold if $a = a(n)$ is given by an inverse polynomial function $a(n) = 1/q(n)$? How about if $q(n) = 2^n$? How slow-growing can $a(n)$ be to make this work?

Answer: To show $\mathrm{PP}_a \subseteq \mathrm{PP}$, let $L \in \mathrm{PP}_a$ via $R(x,y)$ with $|y| = p = p(|x|)$. Compute $N = \lfloor a2^p \rfloor$, OK this needs $p(n)$ bits of a to be computable in *poly*(n) time. Define

$$R'(x,y') = \begin{cases} y' = 1y \wedge R(x,y) & \text{or} \\ y' = 0y \wedge y \geq N, \end{cases}$$

where we are identifying $\{0,1\}^p$ with $0 \dots 2^p - 1$. Then for all $x \in \{0,1\}^n$,

$$x \in L \iff \#y.R(x,y) > N \iff \#y.R(x,y) + 2^p - N > 2^p \iff \#y'.R'(x,y') > 2^p.$$

Since $|y'| = p+1$ which shows $L \in \mathrm{PP}$. This direction allows a to be anything, even 0 (when it shows $\mathrm{NP} \subseteq \mathrm{PP}$), so it "goes arbitrarily low."

For the other direction, let $L \in \mathrm{PP}$ via $R(x,y)$ and $p(n)$. Observe that for any $k \geq 1$, if we define

$$R_k(x,uy) = |u| = k \wedge (u = 1^k \wedge R(x,y)) \vee u \neq 1^k$$

and $R'_k(x,uy) = R(x,y) \wedge u = 1^k$, then for all x,

$$x \in L \iff \Pr_{u,y}[R_k(x,uy)] > 1 - \frac{1}{2^{k+1}} \iff \Pr_{u,y}[R'_k(x,uy)] > \frac{1}{2^{k+1}}.$$

Provided a is between these extremes, we can compute $N = \lfloor a2^{p+k} \rfloor - 2^{p-1}$ and define

$$R_a(x,uy) = (u \in 1^* \wedge R(x,y)) \vee uy < N.$$

This works for any $a = a(n)$ of the form $1/2^{q(n)}$ for any polynomial q, but not for $a = 0$. The meaningful point is that the new bounding polynomial $p'(n)$ is not lower than $p(n) + q(n)$. So one can pad to make the threshold arbitrarily small (or close to 1) but not in terms of the original bounding polynomial p.

(2) Now given $0 < a < b < 1$, define $\mathrm{BPP}_{a,b}$ to be the class of languages L such that for some $p(n)$ and $R(x,y)$ as above, and all x:

$$x \in L \implies \Pr_y[R(x,y)] \geq b;$$
$$x \notin L \implies \Pr_y[R(x,y)] \leq a.$$

(Here I've left tacit that y ranges over $\{0,1\}^{p(|x|)}$.) Show that $\mathrm{BPP}_{a,b} = \mathrm{BPP}$. But now for the real question: Suppose a and b depend on n as in the final part of problem (1). Most in particular, suppose $q(n)$ and $q'(n)$ are polynomials such that $a(n) = 1/q(n)$ and $b(n) = a(n) + 1/q'(n)$. Then when you do $t(n)$-many trials to amplify the success probability, do you get a higher power of $q(n)$ versus $q'(n)$, or are they about the same?

Answer: First, for the simplest way to show $\mathrm{BPP}_{a,b} \subseteq \mathrm{BPP}$, let $L \in \mathrm{BPP}_{a,b}$ via $R(x,y)$ and $p(n)$ and take $c = (a+b)/2$. The construction in problem (1) gives $R'(x,y')$ with $p'(n) = p(n)+1$ such that $\#y'.R'(x,y') = \#y.R(x,y) + 2^p - c2^p$. Thus

$$x \in L \implies \Pr_y[R(x,y)] \geq b \quad\implies\quad \Pr_{y'}[R'(x,y)] \geq \frac{1}{2} + \frac{b-c}{2};$$
$$x \notin L \implies \Pr_y[R(x,y)] \leq a \quad\implies\quad \Pr_{y'}[R'(x,y)] \leq \frac{1}{2} - \frac{c-a}{2};$$

Thus we have a constant displacement from $\frac{1}{2}$ in both cases, so $L \in \mathrm{BPP}$. The converse inclusion $\mathrm{BPP} \subseteq \mathrm{BPP}_{a,b}$ follows from the rest of (1) (with the same value c in place of a) as will follow from what we do next, but let's stay with analyzing the forward direction for the rest of the question. If $a(n) = 1/q(n)$ and $b(n) = a(n) + 1/q'(n)$, then both $(b-c)/2$ and $(c-a)/2$ equal $1/4q'(n)$. As we saw in lecture, amplifying this to a constant separation by majority vote of $t(n)$ trials needs $t(n) = \Theta(q'(n)^2)$—or put another way, amplifying it to have error below $1/2^{r(n)}$ for a given polynomial $r(n)$ takes $t(n) = \Theta(q'(n)^2 r(n))$ trials. This is quadratic in $q'(n)$, and the mapping-to-1/2 idea took $q(n)$ completely out of the picture. Can we do better?

A hint that indeed we can comes from considering the case $q'(n) = q(n)$, i.e., $b = 2a$. Let us do only $B = 1/b$ trials, accepting iff we get at least one "hit." To note one technical point, this means guessing $Y = y_1, \ldots, y_B \in 0,1^{p(n)}$ uniformly at random *with replacement*, meaning some y_i could be repeated—though with B being polynomial and $|0,1^{p(n)}|$ being exponential the difference from sampling without replacement by guessing a subset of size B can safely be ignored. Now the chance of *not* getting a hit from B independent trials, each of success probability at least $1/B$, is

$$\leq (1 - \frac{1}{B})^B \sim \frac{1}{e} = 0.3678794\ldots$$

with quite rapid convergence as B increases. One the other hand, when $x \notin L$ so that the success probability is $\leq a = b/2$, we have that the probability of *not* getting a hit is

$$\geq (1 - \frac{1}{2B})^B = \left((1 - \frac{1}{2B})^{2B}\right)^{1/2} \sim \sqrt{1/e} = 0.60653\ldots$$

Thus $x \in L \implies \Pr_Y[\bigvee_i R(x,y_i)] > 0.6$ while $x \notin L \implies \Pr_Y[\bigvee_i R(x,y_i)] < 0.4$. Thus, in the case $a = 1/q(n)$, we have achieved a constant separation with only $B = \Theta(q(n)) = \Theta(q'(n))$ trials, not $\Theta(q'(n)^2)$ trials as before. For any constants $a < b$ one can get a constant separation by choosing a slightly lower number $B' < 1/b$ of trials, and you may enjoy working out how the ϵ giving the separation of probability $\frac{1}{2} + \epsilon$ from $\frac{1}{2} - \epsilon$ depends on the ratio $\frac{b}{a}$. This anyway is enough to show $\mathrm{BPP} \subseteq \mathrm{BPP}_{a,b}$.

For a pretty much full treatment of the non-constant case $a(n) = 1/q(n)$ and $b(n) = a(n)+1/q'(n)$, let us return to the standard-deviation analysis used in lecture as an alternative to Chernoff bounds, this time for the asymmetric binomial distributions $\mathbf{B}_{t(n),a,1-a}$. Let $c = a + d(b-a)$ where we might choose d different from $1/2$ to allow for our threshold possibly being to a or to b depending on how $q'(n)$ giving $b = a + \frac{1}{q'(n)}$ relates to $q(n) = 1/a$. Our algorithm is to do $t = t(n)$ trials—independently with replacement but again the difference to guessing subsets without replacement is negligible—and accept iff we get at least ct hits. The error conditions we have to bound away are:

- getting ct or more hits from $\mathbf{B}_{t,a,1-a}$ when $x \notin L$;
- getting fewer than ct hits from $\mathbf{B}_{t,b,1-b}$ when $x \in L$.

Addressing the former error, the standard deviation of $\mathbf{B}_{t,a,1-a}$ is $\sigma = \sqrt{ta(1-a)}$, and the proportional standard deviation is $\sigma' = \sigma/t$. To celebrate the fact that CSE696 is currently a physics course in the weeks covering quantum, we will set $1 - a = 1$, so $\sigma' = \sqrt{\frac{a}{t}}$. We want to know what value of t makes $a + \sigma' < c$. This means $\sigma' < c - a = d(b-a)$, so

$$\frac{a}{t} < d^2(b-a)^2.$$

Using our values of a and $b - a$ this becomes

$$\frac{1}{tq(n)} < \frac{d^2}{q'(n)^2},$$

so

$$t > \frac{q'(n)^2}{d^2 q(n)}.$$

Similarly for the second error condition we have $\sigma'' \sim \sqrt{\frac{b}{t}}$ and we want t such that $b - \sigma'' > c$, i.e., $\sigma'' < b - c = (1-d)(b-a)$. We get:

$$\frac{b}{t} < (1-d)^2(b-a)^2,$$

so

$$t > \frac{1/q(n) + 1/q'(n)}{(1-d)^2(1/q'(n))^2} = \frac{1}{(1-d)^2} \cdot \left(\frac{q'(n)^2}{q(n)} + q'(n)\right).$$

If $q'(n) = o(q(n))$, the extra $q'(n)$ term here could make a difference and motivate us to fiddle with d, but in fact the multiplier $\frac{1}{(1-d)^2}$ cannot be made close to zero to offset it. Hence we may as well suppose $d = 1/2$ and drop it out of the asymptotic notation. The upshot is that we always need $\Omega(q'(n))$ trials in order to have a chance of observing any constant separation, and will need more in case $q(n) = o(q'(n))$, namely $\Omega(q'(n) \cdot \frac{q'(n)}{q(n)}$ trials. Or put another way, we save compared to the case $a = 0.5$ when $q(n)$ is sizable so that a is fairly close to zero compared to the separation $b - a$.

Thus the answer is that the powers of $q(n)$ and $q'(n)$ act quite differently, with $q(n)$ being a negative power, sometimes offsetting $q'(n)$ being always quadratic. Whether the savings when $q'(n) \approx q(n)$ is possibly useful is something to file away in one's technical bag-of-tricks—though the real need may be whether it carries over to *extractor*-based improvements to amplification as mentioned briefly in lecture.

(3) Define $\mathcal{U}$ to be the class of languages L such that for some $p(n)$ and $R(x,y)$ as above, and all x,

$$x \in L \iff (\exists! y) R(x,y).$$

The concept to come in section 11.1 is more stringent in requiring L to "promise" that the case where $R(x,y_1)$ and $R(x,y_2)$ hold with $y_1 \neq y_2$ never happens. Here in that case $x \notin L$.

Does $\mathcal{U}$ contain either NP or co-NP? Can you place $\mathcal{U}$ within the second or third level of the polynomial hierarchy? Is $\mathcal{U}$ closed under complements? After answering these warmup questions, show that if $\mathcal{U} \subseteq$ BPP, then NP = RP.

Answer: Suppose $L \in$ co-NP via $x \in L \iff (\forall^p y)\neg R(x,y)$. Define $R'(x,by) = (b = 1 \wedge R(x,y)) \vee by = 0^{p(|x|)+1}$. Then for all x, $x \in L \iff (\exists! y')R'(x,y')$, so $L \in \mathcal{U}$. So co-NP $\subseteq \mathcal{U}$.

For an upper bound, note that if $L \in \mathcal{U}$ via $R(x,y)$ and p, then for all x,

$$x \in L \iff (\exists^p y)R(x,y) \wedge (\forall^p z,z')[R(x,z) \wedge R(x,z') \rightarrow z = z'].$$

A trick here is that we do not have to make either "z" or "z'" the same as "y." Hence the two quantifiers are independent of each other and can be brought out front in either order, which gives $L \in \Sigma_2^p \cap \Pi_2^p$. Best, however, we can "solve" each quantifier by a separate call to an NP oracle, so that

$$L \in \mathrm{P}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}} =_{def} \Delta_2^p.$$

Here the superscripted "[2]" means "with two queries." It is technically important to note that the two queries involved are not "y" and "z' combined with z" but rather strings $0x$ and $1x$ queried to the following combined NP-language:

$$A = \{\, bx : (\exists^p y,z,z')(b = 0 \wedge R(x,y)) \vee (b = 1 \wedge R(x,z) \wedge R(x,z') \wedge z' \neq z) \,\}.$$

Since the query $1x$ is made regardless of the answer to $0x$, this is a 2-*tt* reduction, that is, a "truth-table redction with 2 queries."

On the other hand, there is no evident way to show NP $\subseteq \mathcal{U}$. The community-accepted way to substantiate such a statement is to exhibit an oracle A such that $\mathrm{NP}^A \not\subseteq \mathcal{U}^A$. However, let's leave it as read, and note that in consequence there is no evident way to show that $\mathcal{U}$ is closed under complements (because closure under complements and containment of co-NP implies containment of NP).

But since BPP *is* closed under complements, $\mathcal{U} \subseteq$ BPP does imply NP $\subseteq$ BPP (via co-NP $\subseteq$ BPP). This gives us the hypothesis of the text's Exercise 10.6. In particular, it gives us SAT $\in$ BPP. To get NP $\subseteq$ RP it suffices to infer SAT $\in$ RP. That is, we need to eliminate the possibility of a formula ϕ being unsatisfiable but our randomized algorithm mistakenly halting and saying that it is satisfiable. We do this by demanding that it output a satisfying asignment whenever it says "satisfiable." By amplifying the assumed BPP formulation for SAT we can ensure that the n queries needed by the binary-search algorithm to construct a satisfying assignment all give correct answers with high probability, so that we get the needed assignment. (In fact, we don't need exponentially small error; error $1/n^2$ is plenty.)

(4) Oracle circuits have k-ary *oracle gates* g for arbitrary k (depending on the input length n) such that if $a = a_1 \cdots a_k$ are the binary inputs to g and $A \subseteq \{0,1\}^*$ is the oracle language, then $g(a)$ returns 1 iff $a \in A$. The standard definition of SAT^A uses *oracle clauses* $(u_1, \ldots, u_k)$ with $u_i = \pm a_i$ for each i that are true iff the assignment makes the signed value string of the clause belongs to A. (This is in addition to standard components of Boolean formulas that don't depend on A.) Oracle clauses may be negated. I prefer the somewhat more liberal definition that allows $\pm(u_1, \ldots, u_k)$ to be treated as a literal, just like $\pm w$ for the variable w denoting the output value of an ordinary (NAND) gate. Either way:

(a) Show that SAT^A is NP^A-complete, for any oracle set A.

(b) Define MAJSAT^A and show that it is complete for PP^A, for any A.

It is OK for answers to assume the reader already knows (the NAND-based circuit proof of) the Cook-Levin theorem and to sketch only the essential changes that are needed.

Answer: We can reduce the language of an arbitrary NP^A-machine to the problem of whether there exists y making $C^A(x\#y) = 1$ for some oracle circuit C much as before—a technical details is to program a "guard" gate gadget to govern when the machine has actually submitted the query. The essence of the Cook-Levin proof is then to enforce that the common value w of the output wire(s) from a gate g is correct given the values of the input wires—for oracle gates as well as ordinary NAND gates. Incidentally it is customary to write $A(u_1, \ldots, u_k)$ for both the oracle gates and the oracle clauses, but one needs to keep in mind that "A" is not part of the syntax. It is also possible that a u_ℓ input to the oracle gate could be a negated input variable $\bar{x}_i$ or $\bar{y}_j$, but one could (if desired) insert extra NAND gates computing the identity to "recycle" them as a positively-signed variable. To enforce that the output w of an oracle gate is correct, we can simply write

$$(w \leftrightarrow A(u_1, \ldots, u_k))$$

as a part of the "SAT^A-formula." However, we cannot simply have $A(u_1, \ldots, u_k)$ or $\neg A(u_1, \ldots, u_k)$ be standalone clauses in a *CNF* or *DNF* formula, because the correctness objective would get mixed up with the semantics of A. To define *CNF-SAT*^{A} or *DNF-TAUT*^{A}, it seems we need something like allowing $A(u_1, \ldots, u_k)$ as a *literal*, in clauses of the form

$$(w \vee \bar{A}(u_1, \ldots, u_k)) \wedge (\bar{w} \vee A(u_1, \ldots, u_k)),$$

and similarly for DNF. We can still say that $3\mathrm{SAT}^A$ is complete for NP^A using these "CNF^A" formulas.

We have a similar issue in defining MAJSAT^A, that is do we really want to say $\mathrm{MAJ3SAT}^A$? Either way, it is defined as the corresponding set of "SAT^A-formulas" for which a majority of the assignments are satisfying. The argument that it is PP^A-complete is entirely similar.