CSE696 Spring 2021, Week 5: Bounded-Error Probabilistic Classes

The matrix example makes the probability easy to figure, but it does not show a difference between "polynomial" and "exponential". This is enshrined in the definitions of the complexity classes BPP, RP, and co-RP. It is convenient to think of polynomial-time computable predicates $R(x,y)$ where y ranges over $\{0,1\}^{p(n)}$ with equal length rather than say $|y| \leq p(n)$ (with $n = |x|$ as usual). Then y is a sequence of $p(n)$ coin-flips.

Definition 1: A language A belongs to BPP if there is a polynomial p and a polynomial-time decidable predicate $R(x,y)$ such that for all n and x of length n:

$$\begin{aligned} x \in A &\implies \Pr_{|y|=p(n)}[R(x,y)] > 3/4; \\ x \notin A &\implies \Pr_{|y|=p(n)}[R(x,y)] < 1/4. \end{aligned}$$

If the second probability is always 0 then A is in RP; if instead the first probability is always 0 then A is in co-RP; together these cases are called having *one-sided error*. Note that the first probability being always 1 is equivalent to saying it is always 0 for the complementary predicate $\widetilde{R}(x,y)$, which is where RP and co-RP start to get confusing. The same ability to flip between R and its negation tells right away that BPP is closed under complements, which makes it less confusing. For BPP, we can also combine the conditions into one, namely

$$\Pr_{|y|=p(n)}[A(x) = R(x,y)] > 3/4.$$

But this is often less helpful than having the two separate probabilities. Note that if the second probability is 0 then $R(x,y)$ is impossible when $x \notin A$. It follows that having $R(x,y)$ be true makes y a valid *witness* for $x \in A$, so we have proved the following:

Proposition 1: RP $\subseteq$ NP and co-RP $\subseteq$ co-NP. $\boxtimes$

Of course $L \in$ RP $\iff \widetilde{L} \in$ co-RP, so whether a problem belongs to RP or to co-RP depends on which side one takes as the "yes" side. If you regard $AB = C$ as the yes side and $ABu = Cu$ as the verifying predicate "$R(\langle A,B,C\rangle, u)$", then the matrix example has one-sided error of the "co-RP type", meaning that if the answer is yes then you can never be bluffed into thinking the answer is no; but in a true-negative case there is a tiny chance of getting a false positive (i.e., thinking $AB = C$ because every u that you tried gave $A(Bu) = Cu$). You could say that the language

$$L = \{\langle A,B,C\rangle : AB = C\} \text{ belongs to co-RPTIME}\Big[\widetilde{O}(n^2)\Big],$$

but this notation gets ugly and hides the dependence between the error probability and the time allowed for multiple trials. For polynomial bounds it is even more favorable than for "Oh-tilde" type bounds:

Amplification Lemma 2: If $A \in$ BPP with associated $R(x,y)$ and $p(n)$, then for any polynomial $q(n)$ we can build a polynomial-time decidable $R'(x,z)$ and associated polynomial $p'(n)$ such that for all x,

$$x \in A \implies \Pr_{|z|=p'(n)}[R'(x,z)] > 1 - 2^{-q(n)};$$
$$x \notin A \implies \Pr_{|z|=p'(n)}[R'(x,z)] < 2^{-q(n)}.$$

Moreover, we can achieve this even if the original R and p only give a "non-negligible" advantage, meaning that for some polynomial $r(n) \geq n$,

$$x \in A \implies \Pr_{|y|=p(n)}[R(x,y)] > \frac{1}{2} + \frac{1}{r(n)};$$
$$x \notin A \implies \Pr_{|y|=p(n)}[R(x,y)] < \frac{1}{2} - \frac{1}{r(n)}.$$

Proof Sketch (for now): Regard $z = \langle y_1, y_2, \ldots, y_{q'(n)} \rangle$ where $q'(n) = O(q(n))$ and define $R'(x,z)$ to be the majority vote of the polynomially-many trials $R(x,y_j)$. [The full proof in Arora-Barak comes later in Chapter 7.] ☒

There is a similar amplification lemma for one-sided error; in fact, the details of getting the exponentially small error are simpler because you don't need majority vote. A philosophical point is that the the theoretical software error can be reduced below the chance of hardware error---but when you see something like https://www.wnycstudios.org/podcasts/radiolab/articles/bit-flip (which I heard on NPR last November), maybe that's not so reassuring...

There is also ZPP = RP ∩ co-RP. By the two error-free conditions, this can be characterized as the class of languages (or functions) that have an algorithm A and a polynomial p such that:

- With high probability over random z, $A(x,z)$ halts within $p(|x|)$ steps.
- If and when $A(x,z)$ halts, it always gives correct output.

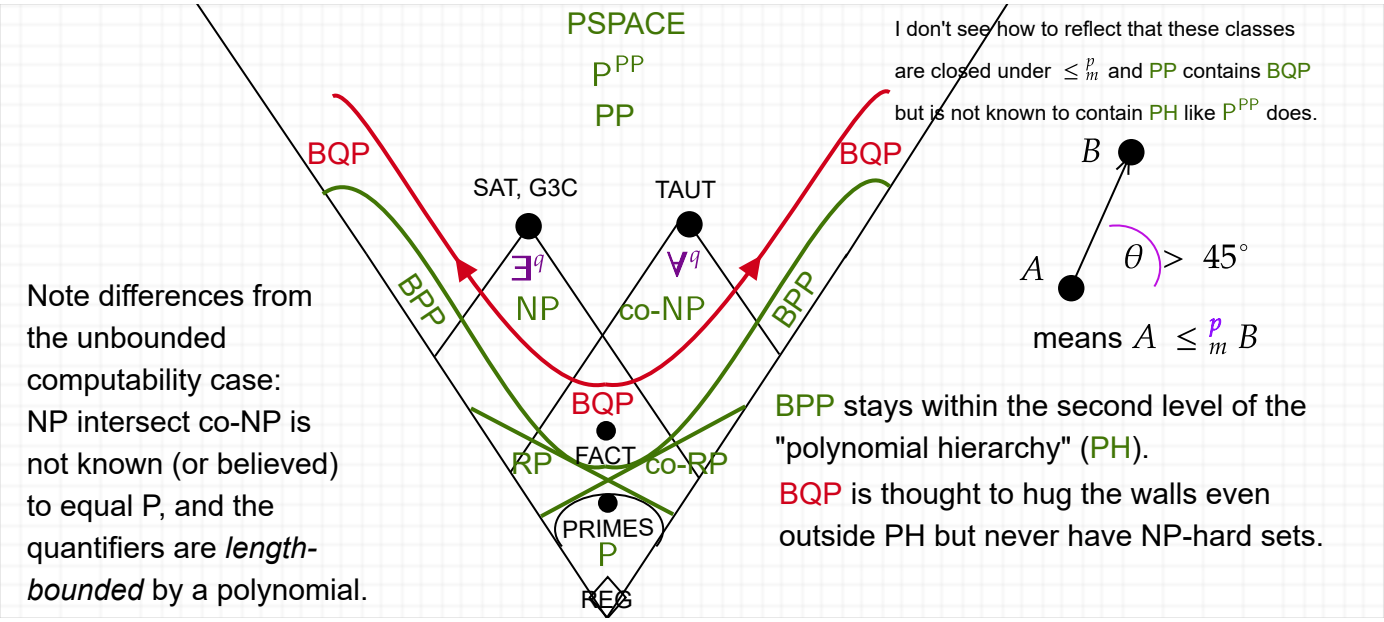
Such an A is often called a **Las Vegas algorithm** to contrast with a **Monte Carlo algorithm** where even after termination there is uncertainty on one or the other side. Again, IMHO the best initial examples are for quasi-linear versus quadratic complexity rather than polynomial versus exponential. In two words: Randomized Quicksort! Another such example should ring a bell:

The (Randomized Greedy Algorithm for the) ***N*-Queens Problem**.

One related family of examples is hash-based storage---where, however, the element of randomness may come from the data rather than the algorithm. Cuckoo Hashing is a particularly nice case.

The definition of the quantum complexity class BQP is similar, except that in place of getting y such that $R(x,y)$ by rolling classical dice, we have a *quantum circuit* C in place of R and get the effect of y by measurements. Amplification and many other properties hold similarly; the main external difference is that the factoring problem and some others belong to BQP but (hopefully!) not to BPP. The

"landscape" of current knowledge is:



The Pivotal Problem [A-B section 7.2.2]

Before 2002, the usual first example of a language in BPP was the language of prime numbers, which was long known to belong to ZPP. That is, before it was **derandomized** by being shown to belong to P. The deterministic algorithm runs with a higher polynomial exponent than the randomized ones, however, so many software primality tests are still randomized. Except for the following bellwether problem, it is hard to find other examples, let alone with two-sided error.

Polynomial Identity Testing (PIT).
Instance: A polynomial formula $f(x_1, \ldots, x_n)$ over $\mathbb{Z}$, $\mathbb{Z}_m$, or a field $\mathbb{F}$ (see notes on degree below).
Question: Does f when "multiplied out" cancel to the all-zero polynomial?

Multiplying out is not so simple---it can take exponential time. Consider how in reducing from **Exactly One 3SAT** to **Binary Linear Equations** (presentation topic 4 of HW6) we get equations $E_1 = 1, E_2 = 1, \ldots, E_m = 1$ from the m clauses. Each equation has 3 variables plus maybe a constant term. We can multiply them together to get a single equation of degree (only!) m:

$$(E_1)(E_2) \cdots (E_m) - 1 = 0.$$

Multiplying this out, however, gives somewhere between 3^m and 4^m terms. Even so, we're hung up on the "NP-side" of looking for one solution, rather than the "co-NP side" of seeing whether all assignments are solutions. This can be done, but you still have to mix in the non-linear equations $x_i^2 - x_i = 0$ to force each variable to be 0 or 1, and even then, the resulting polynomial might not cancel entirely symbolically, as we show with a simple one-variable example next.

A "yes" answer certainly implies that $f(\vec{a}) = 0$ for all arguments $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{F}^n$. Hence if we find an argument $\vec{a} = (a_1, \ldots, a_n)$ such that $f(a_1, \ldots, a_n) \neq 0$, then we know the answer is "no".

1. There are polynomials that are zero on all $\vec{a} \in \mathbb{F}^n$ without multiplying out to zero; a simple one-variable example ($n = 1$) with $p = 2$ is $g(x_1) = x_1^2 - x_1$ over $\mathbb{F}_2$.
2. However, if we enlarge the field to $\mathbb{F}' = \mathbb{F}_4$ or $\mathbb{F}_8$ (etc.) while keeping the mod-2 **characteristic** the same (note those are not the same as the integers mod 4 or mod 8), then $g(x_1)$ is no longer everywhere-zero over $\mathbb{F}'$.
3. Whereas, if $f(x_1, \ldots, x_n)$ multiplies out to zero over $\mathbb{F}$, then it multiplies out to zero over any $\mathbb{F}' \supseteq \mathbb{F}$ of the same characteristic (called an *extension field*), and vice-versa.

Points 1 and 2 are why the fact of **PIT** being in BPP---indeed, with one-sided error like in the $AB = C$ matrix example---does not put **SAT** into BPP. (While composing these notes, I thought of a possible allusion to how working with binary truth values involves the "law of excluded middle" while going to $\mathbb{F}' = \mathbb{F}_4$ or $\mathbb{F}_8$ (etc.) means doing without it---but I am not sure how meaningful it is.)

An important further point is that we can exponentiate the field size with only polynomial work: For any $k > 1$, $\mathbb{F}_{2^k}$ equals the binary vector space $\mathbb{F}_2^k$ augmented with an extra multiplication operation $u*v$ on binary k-tuples. Computing $u*v$ only involves multiplying and dividing by certain single-variable polynomials of degree k modulo 2. With n variables you wind up with (nk)-tuples but the arithmetic involves only $\widetilde{O}(nk)$ work per operation.

The upshot of this is that in stating PIT, we may suppose that the total degree d of the polynomial formula $f(x_1, \ldots, x_n)$ obeys $d < |\mathbb{F}|$. If it doesn't, then we can scale up $\mathbb{F}$ to $\mathbb{F}'$ to make it so---unless the degrees d_n of the formulas f_n for each n are horribly exponential. This allows us to apply the following "strong form" of the **Schwartz-Zippel-(de Millo-Lipton) Lemma**.

Lemma 3: Take any finite subset S of the field $\mathbb{F}$ (if $\mathbb{F}$ is already finite we can just take $S = \mathbb{F}$). Let $f(x_1, \ldots, x_n)$ have total degree at most d. Suppose f does not multiply out to 0. Then

$$\Pr_{a_1,\ldots,a_n \in S}[f(a_1, \ldots, a_n) = 0] \leq \frac{d}{|S|}.$$

There is an alternative weaker form in which d' is the maximum degree in any one of the n variables and the probability conclusion you get is $\leq \frac{d'n}{|S|}$. The weaker form also holds over $\mathbb{Z}$ and $\mathbb{Z}_m$ and other *rings* that are not *fields*. Note that the average degree of a variable is $\frac{d}{n}$ so the numerator $d'n$ is similar to just d, but because this defines d' to be the max, not the average, the result you get is technically weaker (but just as useful in most cases---this is how Debray gives it). Note that I partner with Lipton; I helped him explain at

https://rjlipton.wordpress.com/2009/11/30/the-curious-history-of-the-schwartz-zippel-lemma/

the story of how he and Rich de Millo originally had the weaker form in 1977, a year ahead of Jack Schwartz's stronger form with Richard Zippel in-between. Moreover, I shared an office with Zippel at Cornell for some months in 1986 (if I recall correctly).

Corollary 4: **PIT** (over any of $\mathbb{Z}$, $\mathbb{Z}_m$, or fields $\mathbb{F}$, even infinite fields) belongs to co-RP.

The basic fact underlying the proof is that a *single*-variable polynomial of degree d has at most d roots. The fact of having n variables expands in the denominator and the numerator in a similar manner; formally, this is shown by induction on n. For those interested in the details,

https://nickhar.wordpress.com/2012/02/01/lecture-9-polynomial-identity-testing-by-the-schwartz-zippel-lemma/

also has a nice comparison of **PIT** with the evaluates-to-zero problem. Whether **PIT** belongs to ZPP, let alone to P, is a challenging question.

Perfect Matchings and "Magic" Matrices [compare A-B ch. 7, subsection 7.2.3]

Note that if B is an $n \times n$ matrix with nonnegative entries, then $perm(B) = 0 \implies \det(B) = 0$, because no diagonal products are negative, so the only way the permanent can vanish is when they are all zero, which vanishes the determinant as well. This implication does not necessarily go the other way, the matrix $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ being a simple example. We are interested in cases where this does become an equivalence, so that we can harness the "easy" determinant to do some work of the "hard" permanent function. Now write $A \leqslant B$ if A is obtained from B by zeroing out some entries.

Definition 2: Call an $n \times n$ matrix B with positive rational entries "magic" if for all matrices A such that $A \leqslant B$ we have $\det(A) = 0 \iff perm(A) = 0$.

We can make the above matrix magic simply by changing one of the entries to 2 or to 0.5, say. You would expect me as a responsible teacher to give you an example of a magic $n \times n$ matrix for every value of n, but that is the one thing in this case that humanity does not know how to do. The best we know is the following results.

Theorem 5 [origin unclear]: With vastly high probability, an $n \times n$ matrix with random positive entries of precision $2n \log n$ bits or greater is magic. $\boxtimes$

There are similar theorems about "generic position" for separating hyperplanes in data science; the $n \log n$ comes from Stirling's approximation to $n!$ or just as $\log n^n$, with the extra factor of 2 providing slack to amplify the probability.

Corollary 6: Deciding whether an $n \times n$ bipartite graph G has a perfect matching randomly reduces to computing determinants.

Proof: Generate an $n \times n$ magic matrix B and take A to be the entrywise product of B and the adjacency matrix between the partitions of G, so that $A \leqslant B$. Then G has a perfect matching if and only if $perm(A) = 0$, which by "magic" is if and only if $\det(A) = 0$. $\boxtimes$

The Arora-Barak text gives a more direct argument by László Lovász that applies the S-Z lemma to the determinant polynomial and needs random entries only from the set $\{1, \ldots 2n\}$ to work the "magic" in similar fashion. The general form is more compelling, and also IMHO presents the bottom-story instance of **derandomization**: Can we construct a family of $n \times n$ magic matrices for all n? One might expect Vandermonde and related matrices to fill the bill, but they do not...

Relativizations and Presentations of BPP and RP

We can define a general $\mathsf{BP}[\cdot]$ operator on any complexity class, and do likewise with $\mathsf{RP}[\cdot]$ and $\mathsf{ZPP}[\cdot]$ operators. As with $\mathsf{NP}[\cdot]$ we keep the polynomial length bound but allow any class inside.

Definition 3: For any class C, a language A belongs to $\mathsf{BP}[\mathrm{C}]$ if there is a polynomial p and a predicate $R(x, y)$ in C such that for all n and x of length n:

$$x \in A \implies \Pr_{|y|=p(n)}[R(x,y)] > 3/4;$$
$$x \notin A \implies \Pr_{|y|=p(n)}[R(x,y)] < 1/4.$$

Note, by the way, that if a polynomial-time machine P_j decides $R(x, y)$ then we don't have $A = L(P_j)$. We can associate a nondeterministic machine N_j that on input x guesses a y and accepts x if P_j accepts $\langle x, y\rangle$, but we don't have $A = L(N_j)$ either. Indeed, $L(N_j)$ might equal Σ^* since NTMs accept if *some* y is good, regardless of how few y there are. Instead, we postulate a **probabilistic Turing machine** Q_j that accepts x when the "$> 3/4$" case holds and rejects x when the "$< 1/4$" case holds. In order to call Q_j a **BPP-machine**, we need the **promise** that one or the other case holds for all x. Then, and only then, can we write $A = L(Q_j)$ where the definition of "Q_j accepts x" is that the "$> 3/4$" case holds.

The most important case is $\mathsf{BP}[\mathsf{NP}]$, which we will soon equate with **Arthur-Merlin** protocols when we hit Chapter 8. Now we can make a dangerously misleading definition:

Definition 4: For any oracle language B and class C, the relativization of $\mathsf{BP}[\mathrm{C}]$ to the language B is defined to be $\mathsf{BP}\left[\mathrm{C}^B\right]$. In particular, $\mathsf{BPP}^B = \mathsf{BP}\left[\mathsf{P}^B\right]$, which just means that the $R(x, y)$ predicate belongs to P^B.

Why dangerous? The definition really needs C to be a class of oracle machines, which I started writing $\mathscr{C}$. So BPP^B really leans on the natural presentation of relativized polynomial time by polynomially clocked OTMs $\mathscr{P} = [P_j]$. The nontriviality of this shift from "class of languages" to "collection of machines" shows up right away when we ask:

Open Problem: Does there exist a collection $\mathscr{BPP} = [Q_j]$ of OTMs such that for all oracle languages B, BPP$^B = \left\{L\left(Q_j^B\right)\right\}$?

My use of Q for "quixotic" hints that the answer is ***yo***: yes-and-no. The hitch is that the property in Definition 3 is a **promise property**: If Q_j decides $R(x,y)$, it has the special property that for all x, either the density of "good" strings y is $> 3/4$ or it is $< 1/4$. If this property holds (also) for some nontrivial oracle set B, then either:

- $L\left(Q_j^B\right) =$ the same predicate $R(x,y)$ for all B. In this case, Q_j is called a **robust OTM** and the oracle B is only "helping" the running time, or
- $L\left(Q_j^B\right)$ can be a different predicate R^B for different oracle sets B, but some $R^{B'}$ may lack the $3/4$ versus $1/4$ separation property---whereupon $Q_j^{B'}$ may not define a language in BPP$^{B'}$ after all.

The basic issue even shows up when we merely try to create a recursive presentation of BPP by non-oracle machines. We can do so, but the only way we know how is to use "looking back" to check that the promise has held for shorter inputs $x' = \epsilon, 0, 1, 00, \ldots$ and abort the machine to accept a finite set if and when a violation is revealed. Namely, take our bedrock presentation of P by machines P_j, which we run on inputs $\langle x,y\rangle$. The machine Q_j on input x first spends $n = |x|$ steps running the "looking-back" process. It does not matter that verifying the promise takes exponential time in terms of the earlier strings x' --- looking-back can take any finite time desired in the "long view" while occupying only n steps on any particular input. If the n steps turn up no violation, then Q_j behaves syntactically like the associated NTM N_j which guesses a y and accepts if P_j accepts $\langle x,y\rangle$. If a violation is found, then Q_j rejects---which for x can certainly be regarded as an instance of the "$< 1/4$ of y's" case, indeed with zero y's.

Still in the non-oracle case, we get that either $L(Q_j)$ is finite---and hence belongs to BPP---or that the promise holds for all $x' = \epsilon, 0, 1, 00, \ldots$ and hence did hold for each and every x after all, whereupon Q_j is a genuine BPP machine. For every $A \in$ BPP there is a P_j whose $R(x,y)$ keeps the promise, so that Q_j accepts A under the "$> 3/4$" stipulation. So $[Q_j]$ is a recursive presentation of BPP by machines, so we can apply the uniform diagonalization theorem and etc. to it.

The fly in the ointment is that when the promise is violated for some x', this is only discovered on some much larger string x (indeed, on $x = 0^n$ for some n). For strings of that length and longer, Q_j obeys the BPP promise condition through the "$< 1/4$ of y's" case. But for strings x'' between x' and x, the acceptance criterion for $Q_j(x'')$ is not well-defined. There may well be lots of other undetected

violations on those x''. Therefore, Q_j itself is not a "legal BPP-machine", even though we can argue that however we arbitrarily define membership of these strings x'', the overall language will be finite and hence belong to BPP.

When we have oracles, the situation is compounded by the possibility that every non-robust Q_j may be legal for some oracles and illegal for others. That is, the only legal Q_j may be ones for which acceptance is independent of the oracle. The looking-back idea may work after all for any oracle, but even then it will not give a presentation by "nice" machines.

Where things *really* bite---in the non-oracle and oracle worlds alike---is that even if you add polynomial amounts of padding, the illegality frustrates the idea of building a "universal BPP machine." In particular:

> BPP is not known, and not believed, to have complete sets under polynomial-time Turing reductions---let alone many-one reductions---except for the eventuality that BPP $=$ P. There are oracles C such that BPP^C does not have complete sets even when the reductions may consult C.

This lack in turn impedes diagonalizating against BPP. There is a more-general notion of $\mathrm{BPTIME}[t(n)]$ for general time functions $t(n)$, but whether it has a nontrivial *time hierarchy theorem* has been a hugely thorny question.

Amplification [A-B section 7.4]

BPP Has Small Circuits [A-B section 7.6, skipping 7.5]

BPP is in PH [A-B section 7.7]

PH is in BP[$\oplus$P] and thence in P^{PP} (Toda's Theorem) [A-B section 9.3]