Randomized Complexity Classes; RP

- Let $N$ be a polynomial-time precise NTM that runs in time $p(n)$ and has 2 nondeterministic choices at each step.
- $N$ is a polynomial Monte Carlo Turing machine for a language $L$ if the following conditions hold:
  - If $x \in L$, then at least half of the $2^{p(n)}$ computation paths of $N$ on $x$ halt with “yes” where $n = |x|$.
  - If $x \notin L$, then all computation paths halt with “no.”
- The class of all languages with polynomial Monte Carlo TMs is denoted $\text{RP}$ (randomized polynomial time).\(^{a}\)

\(^{a}\)Adleman and Manders (1977).

Where RP Fits

- $P \subseteq \text{RP} \subseteq \text{NP}$.
  - A deterministic TM is like a Monte Carlo TM except that all the coin flips are ignored.
  - A Monte Carlo TM is an NTM with extra demands on the number of accepting paths.
- $\text{COMPOSITENESS} \in \text{RP}$; $\text{PRIMES} \in \text{coRP};$ $\text{PRIMES} \in \text{RP}$.\(^{a}\)
  - In fact, $\text{PRIMES} \in P$.
- $\text{RP} \cup \text{coRP}$ is another “plausible” notion of efficient computation.

\(^{a}\)Adleman and Huang (1987).

Comments on RP

- Nondeterministic steps can be seen as fair coin flips.
- There are no false positive answers.
- The probability of false negatives, $1 - \epsilon$, is at most 0.5.
- Any constant between 0 and 1 can replace 0.5.
  - By repeating the algorithm $k = \lceil -\frac{1}{\log_2 \frac{1}{1-\epsilon}} \rceil$ times, the probability of false negatives becomes $(1 - \epsilon)^k \leq 0.5$.
- In fact, $\epsilon$ can be arbitrarily close to 0 as long as it is of the order $1/p(n)$ for some polynomial $p(n)$.
  - $\frac{1}{\log_2 1-\epsilon} = O(\frac{1}{x}) = O(p(n))$.

ZPP\(^{a}\) (Zero Probabilistic Polynomial)

- The class ZPP is defined as $\text{RP} \cap \text{coRP}$.
- A language in ZPP has two Monte Carlo algorithms, one with no false positives and the other with no false negatives.
- If we repeatedly run both Monte Carlo algorithms, eventually one definite answer will come (unlike RP).
  - A positive answer from the one without false positives.
  - A negative answer from the one without false negatives.

\(^{a}\)Gill (1977).
The ZPP Algorithm (Las Vegas)

1: \{Suppose \( L \in ZPP \).\}
2: \{\( N_1 \) has no false positives, and \( N_2 \) has no false negatives.\}
3: \textbf{while} \textbf{true} \textbf{do}
4: \textbf{if} \( N_1(x) = \text{“yes”} \) \textbf{then}
5: \textbf{return} \text{“yes”};
6: \textbf{end if}
7: \textbf{if} \( N_2(x) = \text{“no”} \) \textbf{then}
8: \textbf{return} \text{“no”};
9: \textbf{end if}
10: \textbf{end while}

ZPP (concluded)

- The expected running time for the correct answer to emerge is polynomial.
  - The probability that a run of the 2 algorithms does not generate a definite answer is 0.5.
  - Let \( p(n) \) be the running time of each run.
  - The expected running time for a definite answer is
    \[
    \sum_{i=1}^{\infty} 0.5^i p(n) = 2p(n).
    \]
- Essentially, ZPP is the class of problems that can be solved without errors in expected polynomial time.

Et Tu, RP?

1: \{Suppose \( L \in RP \).\}
2: \{\( N \) decides \( L \) without false positives.\}
3: \textbf{while} \textbf{true} \textbf{do}
4: \textbf{if} \( N(x) = \text{“yes”} \) \textbf{then}
5: \textbf{return} \text{“yes”};
6: \textbf{end if}
7: \{But what to do here?\}
8: \textbf{end while}

- You eventually get a “yes” if \( x \in L \).
- But how to get a “no” when \( x \not\in L \)?
- You have to sacrifice either correctness or bounded running time.

PP

- A language \( L \) is in the class \( PP \) if there is a polynomial-time precise NTM \( N \) such that:
  - For all inputs \( x, x \in L \) if and only if more than half of the computations of \( N \) (i.e., \( 2p(n) - 1 + 1 \) or up) on input \( x \) end up with a “yes.”
  - We say that \( N \) decides \( L \) by majority.
- MAJSAT: is it true that the majority of the \( 2^n \) truth assignments to \( \phi \)'s \( n \) variables satisfy it?
- MAJSAT is PP-complete.
- PP is closed under complement.
Large Deviations

- You have a biased coin.
- One side has probability $0.5 + \epsilon$ to appear and the other $0.5 - \epsilon$, for some $0 < \epsilon < 0.5$.
- But you do not know which is which.
- How to decide which side is the more likely—with high confidence?
- Answer: Flip the coin many times and pick the side that appeared the most times.
- Question: Can you quantify the confidence?

The Chernoff Bound

Theorem 67 (Chernoff (1952)) Suppose $x_1, x_2, \ldots, x_n$ are independent random variables taking the values 1 and 0 with probabilities $p$ and $1 - p$, respectively. Let $X = \sum_{i=1}^{n} x_i$. Then for all $0 \leq \theta \leq 1$,

$$\text{prob}[X \geq (1 + \theta)pn] \leq e^{-\theta^2pn/3}.$$

- The probability that the deviate of a binomial random variable from its expected value decreases exponentially with the deviation.
- The Chernoff bound is asymptotically optimal.

*Herman Chernoff (1923–).

The Proof

- Let $t$ be any positive real number.
- Then

$$\text{prob}[X \geq (1 + \theta)pn] = \text{prob}[e^{tX} \geq e^{t(1+\theta)pn}].$$

- Markov’s inequality (p. 399) generalized to real-valued random variables says that

$$\text{prob}[e^{tX} \geq kE[e^{tX}]] \leq 1/k.$$

- With $k = e^{t(1+\theta)pn}/E[e^{tX}]$, we have

$$\text{prob}[X \geq (1 + \theta)pn] \leq e^{-t(1+\theta)pn}E[e^{tX}].$$

The Proof (continued)

- Because $X = \sum_{i=1}^{n} x_i$ and $x_i$’s are independent,

$$E[e^{tX}] = (E[e^{tx_1}])^n = [1 + p(e^t - 1)]^n.$$

- Substituting, we obtain

$$\text{prob}[X \geq (1 + \theta)pn] \leq e^{-t(1+\theta)pn}[1 + p(e^t - 1)]^n \leq e^{-t(1+\theta)pn}e^{pn(e^t-1)}$$

as $(1 + a)^n \leq e^{an}$ for all $a > 0$. 
The Proof (concluded)

• With the choice of $t = \ln(1 + \theta)$, the above becomes
  \[
  \operatorname{prob}[X \geq (1 + \theta)pn] \leq e^{pn[\theta - (1 + \theta)\ln(1 + \theta)]},
  \]

• The exponent expands to
  \[
  -\frac{\theta^2}{2} + \frac{\theta^3}{6} - \cdots
  \]
  for $0 \leq \theta \leq 1$, which is less than
  \[
  -\frac{\theta^2}{2} + \frac{\theta^3}{6} \leq \theta^2 \left(-\frac{1}{2} + \frac{\theta}{6}\right) \leq \theta^2 \left(-\frac{1}{2} + \frac{1}{6}\right) = -\frac{\theta^2}{3}.
  \]

Power of the Majority Rule

From \(\operatorname{prob}[X \leq (1 - \theta)pn] \leq e^{-\frac{\theta^2}{2}pn}\) (prove it):

**Corollary 68** If \(p = (1/2) + \epsilon\) for some \(0 \leq \epsilon \leq 1/2\), then
\[
\operatorname{prob}\left[\sum_{i=1}^{n} x_i \leq n/2\right] \leq e^{-\epsilon^2 n/2}.
\]

• The textbook’s corollary to Lemma 11.9 seems incorrect.

• Our original problem (p. 458) hence demands \(\approx 1.4k/\epsilon^2\) independent coin flips to guarantee making an error with probability at most $2^{-k}$ with the majority rule.

BPP\(^a\) (Bounded Probabilistic Polynomial)

• The class BPP contains all languages for which there is a precise polynomial-time NTM \(N\) such that:
  - If \(x \in L\), then at least $3/4$ of the computation paths of \(N\) on \(x\) lead to “yes.”
  - If \(x \notin L\), then at least $3/4$ of the computation paths of \(N\) on \(x\) lead to “no.”

• \(N\) accepts or rejects by a clear majority.

\(^a\)Gill (1977).

Magic 3/4?

• The number $3/4$ bounds the probability of a right answer away from $1/2$.

• Any constant strictly between $1/2$ and $1$ can be used without affecting the class BPP.

• In fact, $0.5$ plus any inverse polynomial between $1/2$ and $1$,
\[
0.5 + \frac{1}{p(n)},
\]
can be used.
The Majority Vote Algorithm

Suppose $L$ is decided by $N$ by majority $(1/2) + \epsilon$.  
1: for $i = 1, 2, \ldots, 2k + 1$ do 
2: Run $N$ on input $x$; 
3: end for 
4: if “yes” is the majority answer then 
5: “yes”; 
6: else 
7: “no”; 
8: end if

Analysis

- The running time remains polynomial, being $2k + 1$ times $N$’s running time.
- By Corollary 68 (p. 463), the probability of a false answer is at most $e^{-\epsilon^2 k}$.
- By taking $k = \lceil 2/\epsilon^2 \rceil$, the error probability is at most $1/4$.
- As with the RP case, $\epsilon$ can be any inverse polynomial, because $k$ remains polynomial in $n$.

Probability Amplification for BPP

- Let $m$ be the number of random bits used by a BPP algorithm.
  - By definition, $m$ is polynomial in $n$.
- With $k = \Theta(\log m)$ in the majority vote algorithm, we can lower the error probability to $\leq (3m)^{-1}$.

Aspects of BPP

- BPP is the most comprehensive yet plausible notion of efficient computation.
  - If a problem is in BPP, we take it to mean that the problem can be solved efficiently.
  - In this aspect, BPP has effectively replaced P.
- $(\text{RP} \cup \text{coRP}) \subseteq (\text{NP} \cup \text{coNP})$.
- $(\text{RP} \cup \text{coRP}) \subseteq \text{BPP}$.
- Whether BPP $\subseteq (\text{NP} \cup \text{coNP})$ is unknown.
- But it is unlikely that NP $\subseteq$ BPP (p. 483 and p. 765).
coBPP

- The definition of BPP is symmetric: acceptance by clear majority and rejection by clear majority.
- An algorithm for $L \in \text{BPP}$ becomes one for $\overline{L}$ by reversing the answer.
- So $\overline{L} \in \text{BPP}$ and $\text{BPP} \subseteq \text{coBPP}$.
- Similarly $\text{coBPP} \subseteq \text{BPP}$.
- Hence $\text{BPP} = \text{coBPP}$.
- This approach does not work for RP.
- It did not work for NP either.

"The Good, the Bad, and the Ugly"

Circuit Complexity

- Circuit complexity is based on boolean circuits instead of Turing machines.
- A boolean circuit with $n$ inputs computes a boolean function of $n$ variables.
- By identify true with 1 and false with 0, a boolean circuit with $n$ inputs accepts certain strings in $\{0,1\}^n$.
- To relate circuits with arbitrary languages, we need one circuit for each possible input length $n$. 
Formal Definitions

• The **size** of a circuit is the number of **gates** in it.

• A **family of circuits** is an infinite sequence 
  \( C = (C_0, C_1, \ldots) \) of boolean circuits, where \( C_n \) has \( n \) boolean inputs.

• \( L \subseteq \{0, 1\}^* \) has **polynomial circuits** if there is a family of circuits \( C \) such that:
  - The size of \( C_n \) is at most \( p(n) \) for some fixed polynomial \( p \).
  - For input \( x \in \{0, 1\}^* \), \( C_{|x|} \) outputs 1 if and only if \( x \in L \).
    * \( C_n \) accepts \( L \cap \{0, 1\}^n \).

Exponential Circuits Contain All Languages

• Theorem 15 (p. 156) implies that there are languages that cannot be solved by circuits of size \( 2^n/(2n) \).

• But exponential circuits can solve all problems.

**Proposition 69** All decision problems (decidable or otherwise) can be solved by a circuit of size \( 2^{n+2} \).

• We will show that for any language \( L \subseteq \{0, 1\}^* \), \( L \cap \{0, 1\}^n \) can be decided by a circuit of size \( 2^{n+2} \).

The Circuit Complexity of P

**Proposition 70** All languages in P have polynomial circuits.

• Let \( L \in P \) be decided by a TM in time \( p(n) \).

• By Corollary 28 (p. 242), there is a circuit with \( O(p(n)^2) \) gates that accepts \( L \cap \{0, 1\}^n \).

• The size of the circuit depends only on \( L \) and the length of the input.

• The size of the circuit is polynomial in \( n \).
Languages That Polynomial Circuits Accept

• Do polynomial circuits accept only languages in P?

• There are undecidable languages that have polynomial circuits.
  – Let $L \subseteq \{0,1\}^*$ be an undecidable language.
  – Let $U = \{1^n : \text{the binary expansion of } n \text{ is in } L\}$.
  – $U$ must be undecidable.
  – $U \cap \{1\}^n$ can be accepted by $C_n$ that is trivially false if $1^n \not\in U$ and trivially true if $1^n \in U$.
  – The family of circuits $(C_0, C_1, \ldots)$ is polynomial in size.

A Patch

• Despite the simplicity of a circuit, the previous discussions imply the following:
  – Circuits are not a realistic model of computation.
  – Polynomial circuits are not a plausible notion of efficient computation.

• What gives?

• The effective and efficient constructibility of $C_0, C_1, \ldots$.

Uniformity

• A family $(C_0, C_1, \ldots)$ of circuits is uniform if there is a log $n$-space bounded TM which on input $1^n$ outputs $C_n$.
  – Circuits now cannot accept undecidable languages (why?).
  – The circuit family on p. 478 is not constructible by a single Turing machine (algorithm).

• A language has uniformly polynomial circuits if there is a uniform family of polynomial circuits that decide it.

Uniformly Polynomial Circuits and P

Theorem 71 $L \in P$ if and only if $L$ has uniformly polynomial circuits.

• One direction was proved in Proposition 70 (p. 477).

• Now suppose $L$ has uniformly polynomial circuits.

• Decide $x \in L$ in polynomial time as follows:
  – Let $n = |x|$.
  – Build $C_n$ in log $n$ space, hence polynomial time.
  – Evaluate the circuit with input $x$ in polynomial time.

• Therefore $L \in P$. 

Assume $n$’s leading bit is always 1 without loss of generality.
Relation to P vs. NP

- Theorem 71 implies that P ≠ NP if and only if NP-complete problems have no uniformly polynomial circuits.
- A stronger conjecture: NP-complete problems have no polynomial circuits, uniformly or not.
- The above is currently the preferred approach to proving the P ≠ NP conjecture—without success so far.
  - Theorem 15 (p. 156) states that there are boolean functions requiring \(2^n/(2n)\) gates to compute.
  - In fact, almost all boolean functions do.

The Proof

- Let \(L \in \text{BPP}\) be decided by a precise NTM \(N\) by clear majority.
- We shall prove that \(L\) has polynomial circuits \(C_0, C_1, \ldots\).
- Suppose \(N\) runs in time \(p(n)\), where \(p(n)\) is a polynomial.
- Let \(A_n = \{a_1, a_2, \ldots, a_m\}\), where \(a_i \in \{0, 1\}^{p(n)}\).
- Let \(m = 12(n + 1)\).
- Each \(a_i \in A_n\) represents a sequence of nondeterministic choices—i.e., a computation path—for \(N\).

BPP’s Circuit Complexity

**Theorem 72 (Adleman (1978))** All languages in BPP have polynomial circuits.

- Our proof will be nonconstructive in that only the existence of the desired circuits is shown.
  - Something exists if its probability of existence is nonzero.
- How to efficiently generate circuit \(C_n\) given \(1^n\) is not known.
- If the construction of \(C_n\) is efficient, then P = BPP, an unlikely result.

The Proof (continued)

- Let \(x\) be an input with \(|x| = n\).
- Circuit \(C_n\) simulates \(N\) on \(x\) with each sequence of choices in \(A_n\) and then takes the majority of the \(m\) outcomes.
- Because \(N\) with \(a_i\) is a polynomial-time TM, it can be simulated by polynomial circuits of size \(O(p(n)^2)\).
  - See the proof of Proposition 70 (p. 477).
- The size of \(C_n\) is therefore \(O(mp(n)^2) = O(np(n)^2)\), a polynomial.
- We next prove the existence of \(A_n\) making \(C_n\) correct.
The Proof (continued)

• Call \( a_i \) **bad** if it leads \( N \) to a false positive or a false negative answer.

• Select \( A_n \) *uniformly randomly*.

• For each \( x \in \{0, 1\}^n \), 1/4 of the computations of \( N \) are erroneous.

• Because the sequences in \( A_n \) are chosen randomly and independently, the expected number of bad \( a_i \)'s is \( m/4 \).

• By the Chernoff bound (p. 459), the probability that the number of bad \( a_i \)'s is \( m/2 \) or more is at most

\[
e^{-m/12} < 2^{-(n+1)}.
\]

The Proof (concluded)

• The error probability is \( < 2^{-(n+1)} \) for each \( x \in \{0, 1\}^n \).

• The probability that there is an \( x \) such that \( A_n \) results in an incorrect answer is \( < 2^n 2^{-(n+1)} = 2^{-1} \).

\[- \text{prob}[A \cup B \cup \cdots] \leq \text{prob}[A] + \text{prob}[B] + \cdots.\]

• So with probability one half, a random \( A_n \) produces a correct \( C_n \) for *all* inputs of length \( n \).

• Because this probability exceeds 0, an \( A_n \) that makes majority vote work for all inputs of length \( n \) exists.

• Hence a correct \( C_n \) exists.