RANDOMIZED COMPUTATION

- Monte Carlo algorithms and random walks
- Randomized complexity classes: RP, ZPP, PP, BPP
- Perfect and slightly random sources
- Classes $\delta$-RP, $\delta$-BPP
- Circuit complexity

(C. Papadimitriou: Computational complexity, Chapter 11)

Perfect matching—cont’d

- This is related to computing determinants of a matrix:
  Given a graph $G$, construct an $n \times n$ matrix $A_G$ where the element $i, j$ is a variable $x_{ij}$ iff $(u_i, v_j) \in E$ otherwise 0.

$$\det A_G = \sum_\pi \sigma(\pi) \prod_{i=1}^n A_{i, \pi(i)}$$

where $\pi$ ranges over permutations of $n$ elements and each term is of the form

$$\sigma(\pi) a_{i_1, \pi(1)} \cdots a_{i_n, \pi(n)}$$

- Hence, $G$ has a perfect matching iff $\det A_G$ is not identically 0.
- Testing whether $\det A_G$ is identically 0 can be done using a randomized algorithm.

Randomized algorithm for perfect matching

Given a matrix $A^G(x_1, \ldots, x_m)$ with $m$ variables:

Choose $m$ random integers $i_1, \ldots, i_m$ (between 0 and $M = 2m$)
Compute $\det A^G(i_1, \ldots, i_m)$ (by Gaussian elimination).
If $\det A^G(i_1, \ldots, i_m) \neq 0$, then return “$G$ has a perfect matching”
If $\det A^G(i_1, \ldots, i_m) = 0$, then return “$G$ probably has no perfect matching”

Properties:
- No false positives (if “yes” is returned, this is correct).
- False negatives possible (if “no” is returned, this might be wrong).
Monte Carlo algorithm

➤ Polynomial randomized algorithm
➤ No false positives
➤ The probability of false negatives no more than $\frac{1}{2}$.
☞ The previous algorithm is a Monte Carlo algorithm for perfect matching: it can be shown that the probability of false negatives is no more than $\frac{1}{2}$ when the integers are randomly selected between 0 and $2m$.
☞ If the probability of false negatives is $\varepsilon > \frac{1}{2}$, we can perform $k$ independent experiments and the probability of false negatives is reduced to $\varepsilon^k$ (and running time remains polynomial).

Random Walks—cont’d

➤ A randomized walk algorithm for SAT:
   Take any truth assignment $T$ and repeat $r$ times:
   If there is not unsatisfiable clauses, return “satisfiable”
   Otherwise take any unsatisfiable clause
   Pick any of its literals at random and flip it in $T$.
   After $r$ repetition return “probably unsatisfiable”
➤ Is this a Monte Carlo algorithm?
   No false positives but the probability of false negatives is high!
   (An exponential number of repetitions $r$ is needed to achieve low probability for classes of 3SAT problems).

Monte Carlo algorithm for composite

➤ Fermat’s Theorem: For a prime $N$, for all $0 < a < N$, $a^{N-1} = 1 \mod N$.
➤ Fermat test for COMPOSITE:
   Pick random residue $a$ modulo $N$.
   If $a^{N-1} \neq 1 \mod N$, then return “$N$ is composite”
   Otherwise answer “$N$ is probably prime”
➤ Monte Carlo algorithm?
   • By Fermat’s Theorem no false positive.
   • But is it the case that for a composite, for at least half of its nonzero residues $a$, $a^{N-1} \neq 1 \mod N$?
     (No, Carmichael numbers are exceptions)
Monte Carlo algorithm for composite—cont’d

➤ A refined algorithm for testing compositeness of \( N \)

Generate a random integer \( M \) between 2 and \( N - 1 \);
If \( (M, N) > 1 \) then return “\( N \) is a composite”
else
  if \( (M, N) \neq M^{N-1} \mod N \) then return “\( N \) is a composite”
  else return “\( N \) is probably a prime”.

where \( (M, N) \) is the greatest common divisor of \( M \) and \( N \)
and \( (M | N) \) is the Jacobi symbol.

➤ This is a Monte Carlo algorithm:

\( (M, N) \) and \( (M | N) \) can be computed in polynomial time, no false positives and the probability of false negative at most \( \frac{1}{2} \).

The class \( \text{RP} \)

Definition. Let \( L \) be a language. A polynomial time Monte Carlo Turing machine for \( L \) is a nondeterministic Turing machine \( N \)
(i) which is precise having exactly two nondet. choices at each step;
(ii) the number of steps in each computation for an input of length \( n \)
is \( p(n) \), a polynomial and
(iii) for each input \( x \):
  • If \( x \in L \), then at least half of the \( 2^{p(|x|)} \) computations of \( N \) on \( x \)
halt with “yes”.
  • If \( x \notin L \), then all the \( 2^{p(|x|)} \) computations halt with “no”.

The class of all languages with polynomial time Monte Carlo Turing machines is denoted by \( \text{RP} \) (randomized polynomial time).

Randomized complexity classes

➤ Randomized algorithms (such as Monte Carlo ones) can be analyzed using nondeterministic Turing machines but with a different interpretation of what it means for such a machine to accept its input.

➤ No coin-flipping is needed in the Turing machine!
The class RP—cont’d

The power of RP would not be affected if the probability of acceptance were not $\frac{1}{2}$ but any number $0 < \varepsilon < 1$:

- If $\varepsilon < \frac{1}{2}$, “repeat” the algorithm $k$ times and accept iff at least one of the $k$ computations accepts otherwise reject.
- Now the probability of false negative is at most $(1 - \varepsilon)^k$.
- By taking $k = \left\lceil -\frac{1}{\log(1-\varepsilon)} \right\rceil$, the probability of false negative is at most $\frac{1}{2}$.
- The running time is $k$ times the original.
- As $\frac{1}{\log(1-\varepsilon)} \approx \frac{1}{\varepsilon}$, $\varepsilon$ could even be of the form $\frac{1}{p(n)}$ where $p(n)$ is a polynomial and the overall algorithm would remain polynomial.

The class ZPP

- coRP: the languages having Monte Carlo machines with no false negatives and a limited number of false positives.
- PRIMES in coRP
- $ZPP = RP \cap coRP$ is the class of languages with Las Vegas algorithms (polynomial randomized algorithms with zero probability of error).
- A Las Vegas algorithm = two Monte Carlo algorithms: one for the language and one for its complement.
- Running $k$ independent experiments with both algorithms:
  (i) sooner or later a definite answer will come: either a positive answer from the algorithm with no false positives or a negative one from the algorithm with no false negatives.
  (ii) probability of a definite answer is at least $1 - 2^{-k}$.
- PRIMES in RP and thus in ZPP.

The class PP

- Consider the problem MAJSAT:
  Given a Boolean expression, is it true that the majority of the $2^n$ truth assignments to its $n$ variables satisfy it.
- It is not clear that MAJSAT is in NP (and thus less likely in RP).
- PP is the class of languages $L$ having a nondeterministic polynomially bounded Turing machine $N$ (precise and with two choices each step) such that for all inputs $x$, $x \in L$ iff more than half of the computations of $N$ on input $x$ end up accepting.

Theorem. MAJSAT is PP-complete.

Theorem. NP ⊆ PP.

PP is closed under complement.
The class PP—cont’d

- \( \text{ZPP} \subseteq \text{RP} \subseteq \text{NP} \subseteq \text{PP} \)
- \( \text{ZPP, RP} \) are plausible notions of efficient randomized computations (but \( \text{PP} \) is not).
- \( \text{PP} \) cannot be used algorithmically because acceptance by majority is too fragile: the acceptance probability can be \( \frac{1}{2} + 2^{-n|x|} \) and there is no plausible efficient experimentation that can detect such accepting behaviour (see below).

Detecting the more likely side of a bias coin

- To understand this consider the following problem:
  You have a biased coin with one side having probability \( \frac{1}{2} + \varepsilon \) and the other \( \frac{1}{2} - \varepsilon \). How to detect which side is more likely?
  Solution: Flip the coin many times and pick the side that appeared the most times. But how many times?

  - The Chernoff bound:
    Suppose that \( x_1, \ldots, x_n \) are independent random variables taking the values 1 and 0 with probabilities \( p \) and \( p - 1 \), respectively, and consider their sum \( X = \sum_{i=1}^{n} x_i \). Then for all \( 0 \leq \theta \leq 1 \),
    \[ \text{prob}[X \geq (1 + \theta)pn] \leq e^{-\frac{\theta^2}{2}}pn. \]
    - The probability that \( X \) deviates from its expected value (\( pn \)) decreases exponentially with the deviations.

Corollary:

- If \( p = \frac{1}{2} + \varepsilon \) for some \( \varepsilon > 0 \), then \( \text{prob}[\sum_{i=1}^{n} x_i \geq \frac{n}{2}] \leq e^{-\frac{\varepsilon^2}{6}}n. \)
- A bias of \( \varepsilon \) can be detected with reasonable confidence by taking a majority of about \( \frac{1}{\varepsilon} \) experiments (\( e^{-\frac{\varepsilon^2}{6\varepsilon^2}} = 0.85 \)).
- For a \( \text{PP} \) problem the bias \( \varepsilon \) can be as small as \( 2^{-n|x|} \): an exponential number of repetitions of the algorithm is required to determine the correct answer with reasonable confidence.

- Is there some plausible notion of realistic computation between \( \text{RP} \) and \( \text{PP} \)?

\( \mathbb{BPP} \)

The class BPP

- \( \mathbb{BPP} \) is the class of languages \( L \) having a nondeterministic polynomially bounded Turing machine \( N \) (precise and with two choices each step) such that for all inputs \( x \),
  - if \( x \in L \), then at least \( \frac{1}{2} \) of the computations of \( N \) on \( x \) accept;
  - if \( x \not\in L \), then at least \( \frac{1}{2} \) of the computations of \( N \) on \( x \) reject;
  (bounded probability of error)
- \( \text{RP} \subseteq \mathbb{BPP} \subseteq \text{PP} \).
- Open: \( \mathbb{BPP} \subseteq \text{NP} \).
- \( \mathbb{BPP} \) is closed under complement.
- Semantic class
- No known complete problem
**Random Sources**

- In order to implement randomized algorithms (e.g., those for RP and BPP), we need a source of random bits.
- A perfect random source is a random variable with values that are infinite sequences \((x_1, x_2, \ldots)\) of bits such that for all \(n > 0\) and for all \((y_1, y_2, \ldots, y_n) \in \{0,1\}^n\)
  \[
  \text{prob}[x_i = y_i, i = 1, \ldots, n] = 2^{-n}
  \]
- A Monte Carlo algorithm could be implemented using a random source by generating a sequence \((x_1, x_2, \ldots)\) of bits and choosing the transition at each step \(i\) according to the bit \(x_i\).

**Random Sources—cont’d**

- A problem: where to find a perfect random source?
- A perfect random source should be
  - independent: the probability that \(x_i = 1\) does not depend on the previous or future outcomes
  - fair: the probability that \(x_i = 1\) should be exactly \(\frac{1}{2}\).
- The important requirement is independence:
  Any independent but unfair random sequence of bits can be turned into a fair one as follows:
  (i) Break the sequence in pairs and
  (ii) interpret: 01 \(\sim\) 0, 10 \(\sim\) 1 (ignoring 00 and 11).

**Slightly Random Sources**

- Perfect random sources seem to be hard to implement physically.
- A weaker concept: \(\delta\)-random source
  Let \(\delta\) be a number \(0 < \delta < \frac{1}{2}\) and \(p\) any function \(\{0,1\}^* \mapsto [\delta, 1 - \delta]\) (a highly complex function unknown to us).
  The \(\delta\)-random source \(S_p\) is a random variable with infinite bit sequences as values where the probability that the first \(n\) bits have the values \(y_1, y_2, \ldots, y_n\) is
  \[
  \prod_{i=1}^{n} (y_i p(y_1 \ldots y_{i-1}) + (1 - y_i)(1 - p(y_1 \ldots y_{i-1})))
  \]
  (Notice: the probability that the \(i\)th bit is 1 is \(p(y_1 \ldots y_{i-1})\), a number between \(\delta\) and \(1 - \delta\) that depends in an arbitrary way on all previous outcomes \(y_1 \ldots y_{i-1}\).)
Slightly random sources—cont’d

➤ Now $\delta \leq p(y_1 \ldots y_{i-1}) \leq 1 - \delta$
➤ A $\frac{1}{2}$-random source is a perfect random source.
➤ A $\delta$-random source with $\delta < \frac{1}{2}$ is a slightly random source.
➤ Slightly random sources: Geiger counters, Zehner diodes, coins

The classes $\delta$-RP and $\delta$-BPP—cont’d

➤ Let $N$ be a precise, polynomially bounded nondeterministic Turing Machine with exactly two choices per step.
➤ On input $x$ the computation $N(x)$ is in effect a full binary tree of depth $n = p(|x|)$ (having $2^{n+1} - 1$ nodes of which $2^n$ are leaves and $2^n - 1$ internal).
➤ Let $\delta$ be a number $0 < \delta < \frac{1}{2}$. A $\delta$-assignment $F$ is a mapping from the set of edges of $N(x)$ to the interval $[\delta, 1 - \delta]$ such that the two edges leaving each internal node are assigned numbers adding up to one ($p$ is precisely $F$ on 1-choices).

The classes $\delta$-RP and $\delta$-BPP

➤ Given a $\delta$-assignment $F$ for each leaf $l$, $\text{prob}[l] = \Pi_{a \in P(l)} F(a)$ where $P(l)$ is the path from the root to leaf $l$.
➤ $\text{prob}[N(x) = \text{"yes"}|F]$ is the sum of $\text{prob}[l]$ for all "yes" leaves $l$ of $N(x)$.
➤ We say that a language $L$ is in $\delta$-RP if there is a nondeterministic machine $N$, standardized as above, such that if $x \in L$, then $\text{prob}[N(x) = \text{"yes"}|F] \geq \frac{1}{2}$ and if $x \not\in L$, then $\text{prob}[N(x) = \text{"yes"}|F] = 0$ for all $\delta$-assignments $F$.
➤ A language $L$ is in $\delta$-BPP if there is a nondeterministic machine $N$ such that if $x \in L$, then $\text{prob}[N(x) = \text{"yes"}|F] \geq \frac{1}{2}$ and if $x \not\in L$, then $\text{prob}[N(x) = \text{"no"}|F] \geq \frac{3}{4}$ for all $\delta$-assignments $F$. 
Example. Consider the computation tree and 0.1-assignment

\[ \text{prob}[L] = \prod_{a \in P(L)} F(a) = 0.6 \cdot 0.9 \cdot 0.8 = 0.432 \]

and for the right most leaf \( R \)
\[ \text{prob}[R] = \prod_{a \in P(R)} F(a) = 0.4 \cdot 0.5 \cdot 0.6 = 0.120 \]

Simulating a randomized algorithm

- Assume that \( L \in \text{BPP} \), i.e., \( L \) is decided by a NTM \( N \) by clear majority.
- Construct a machine \( N' \) deciding \( L \) by clear majority when driven by any slightly random source.
- The basic idea: confuse the “adversary” by shattering the slightly random bits using inner products.
- Inner product of two sequences of bits \( \kappa = (\kappa_1, \ldots, \kappa_k) \) and \( \lambda = (\lambda_1, \ldots, \lambda_k) \) is the bit obtained by \( \kappa \cdot \lambda = \sum_{i=1}^{k} \kappa_i \lambda_i \mod 2 \).

Simulating machine \( N' \)

- On input \( x \), let \( n = p(|x|) \) be the length of \( N' \)’s computation on \( x \) and let \( k \) be an integer (a parameter depending on \( n \) and \( \delta \)).
- Generate \( n \) sequences of bits (blocks) \( \beta_1, \ldots, \beta_n \) using a \( \delta \)-random source where each \( \beta_i \) contains \( k \) bits.
- Do \( 2^k \) parallel simulations of \( N \) with the sequences of choices
\[ T = \{(\beta_1, \ldots, \beta_n) : \kappa = 0, 1, \ldots, 2^k - 1 \} \]
\[ = \{(\beta_1 \cdot 0, \ldots, \beta_n \cdot 0), \ldots, (\beta_1 \cdot (2^k - 1), \ldots, \beta_n \cdot (2^k - 1))\} \]
- Of the \( 2^k \) answers, adopt the majoritanian one as the answer of \( N' \).
Simulating machine $N'$ — cont’d

To reduce the probability of a false answer by $N'$ to at most $1/4$:

➤ Assume that probability of wrong answer by $N$ is $1/32$ (instead of $1/4$).
➤ Set $k = \lceil \log_2 n + 5 \rceil$
➤ Now $N'$ works within time $O(n^2k) = O(n^{1+\frac{1}{32-2\delta^2}})$, i.e. in polynomial time.

Corollary. For any $\delta > 0$, $\delta$-RP = RP.

CIRCUIT COMPLEXITY

➤ A Boolean circuit with $n$ inputs accepts a string $x = x_1 \ldots x_n$ of length $n$ in $\{0, 1\}^*$ iff the output of the circuit is true given $x$ as its input (i.e., input $i$ is true if $x_i$ is 1 and false otherwise).
➤ To relate circuits to strings of arbitrary length families of circuits are introduced.
➤ 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 $n$ input variables.

Languages with polynomial circuits

➤ A language $L \subseteq \{0, 1\}^*$ has polynomial circuits if there is a family $C = (C_0, C_1, \ldots)$ such that
  (i) the size of $C_n$ is at most $p(n)$ for some fixed polynomial $p$;
  (ii) for all $x \in \{0, 1\}^*$, $x \in L$ iff the output of $C_{|x|}$ is true when $x$ is given as input to $C_{|x|}$.

Proposition. All languages in $\mathsf{P}$ have polynomial circuits.

Proof. By the $\mathsf{P}$-completeness proof of CIRCUIT VALUE:

Given a Turing machine $M$, its input $x$, and running time $p(|x|)$, a variable-free polynomial size circuit $C(M, x, p)$ is constructed such that the output of $C(M, x, p)$ is true iff $M$ accepts $x$.

It is easy to modify input gates of $C(M, x, p)$ to variable gates reflecting the symbols in $x$.

Languages with polynomial circuits

Proposition. There are undecidable languages that have polynomial circuits.

Proof.

➤ Let $L \subseteq \{0, 1\}^*$ be an undecidable language and let $U \subseteq \{1\}^*$ be $U = \{1^n : \text{the binary expansion of } n \text{ is in } L\}$.
➤ $U$ is undecidable.
➤ $U$ has polynomial circuits $(C_0, C_1, \ldots)$ where $C_n$ consists of $n$ input gates and
  • n-1 AND-gates (conjunction of all inputs) if $1^n \in U$ and
  • false output gate if $1^n \notin U$. 
Uniformly polynomial circuits

A family $C = (C_0, C_1, \ldots)$ of circuits is said to be uniform if there is a $\log n$-space bounded Turing machine which on input $1^n$ outputs $C_n$.

A language $L \subseteq \{0, 1\}^*$ has uniformly polynomial circuits if there is a uniform family of polynomial circuits $C = (C_0, C_1, \ldots)$ that decides $L$.

Theorem. A language $L$ has uniformly polynomial circuits iff $L \in \mathbf{P}$.

Proof. ($\Rightarrow$) $x \in L$ can be decided in polynomial time by constructing $C_{|x|}$ in $\log |x|$ space (and hence in polynomial time) and evaluating it for input $x$ (in polynomial time).

($\Leftarrow$) By the $\mathbf{P}$-completeness proof of CIRCUIT VALUE: Given a Turing machine $M$, its input $x$, and running time $p(|x|)$, the circuit $C(M, x, p)$ can be constructed in $\log |x|$ space.

Polynomial circuits and $P$ vs $NP$

$P \neq NP$ equivalent to

Conjecture A: $NP$-complete problems have no uniformly polynomial circuits.

Conjecture B: $NP$-complete problems have no polynomial circuits, uniform or not.

Most Boolean functions do not have small circuits.

An approach to establish $P \neq NP$: show Conjecture B for some $NP$-complete problem.

Cannot be used for establishing $P \neq BPP$

Theorem. All languages in $BPP$ have polynomial circuits.

Learning Objectives

- The concepts of Monte Carlo and Las Vegas algorithms
- The key randomized complexity classes: $RP, ZPP, PP, BPP$
- The concepts of perfect and slightly random sources
- Basic concepts of circuit complexity: languages with polynomial circuits and uniformly polynomial circuits.