Advanced Computational Complexity

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 600

Subsection 1

Probabilistic Turing Machines

Randomized Algorithms

- A randomized algorithm is an algorithm that may involve random choices.

  Example: We may initialize a variable with an integer chosen at random from some range.
- In practice randomized algorithms are implemented using a random number generator.
- It turns out that it suffices to have a random number generator that generates random bits, i.e., produces the bit 0 with probability $\frac{1}{2}$ and the bit 1 with probability $\frac{1}{2}$.
- We say such generators "toss fair coins".
- To model randomized algorithms we use probabilistic Turing machines (PTMs).

Probabilistic Turing Machines

Definition (Probabilistic Turing Machine)

A **probabilistic Turing machine (PTM)** is a Turing machine with two transition functions δ_0, δ_1.

- To execute a PTM M on an input x, we choose in each step:
  - With probability $\frac{1}{2}$ to apply the transition function δ_0;
  - With probability $\frac{1}{2}$ to apply the transition function δ_1.

  This choice is made independently of all previous choices.
- The machine only outputs 1 ("Accept") or 0 ("Reject").

  We denote by $M(x)$ the random variable corresponding to the value M writes at the end of this process.
- For a function $T : \mathbb{N} \to \mathbb{N}$, we say that M **runs in $T(n)$-time** if, for any input x, M halts on x within $T(|x|)$ steps, regardless of the random choices it makes.

Comparing PTMs with NDTMs

- A nondeterministic TM is also a TM with two transition functions.
- Hence, PTMs and NTMs are syntactically similar.
- The difference is in how we interpret the working of the TM.
- Consider PTMs:
  - In a PTM, each transition is taken with probability $\frac{1}{2}$.
  - A computation that runs for time t gives rise to 2^t branches in the graph of all computations, each of which is taken with probability $\frac{1}{2^t}$.
  - $\Pr[M(x) = 1]$ is simply the fraction of branches that end with M outputting a 1.
- An NDTM is said to accept the input if there exists a branch that outputs 1, whereas in the case of a PTM we consider the fraction of branches for which this happens.
- PTMs and NDTMs are also different in terms of intention.
- PTMs, like deterministic TMs and unlike NDTMs, are intended to model realistic computation devices.

The classes BPTIME and BPP

- The class BPP aims to capture efficient probabilistic computation.
- For a language $L \subseteq \{0,1\}^*$ and $x \in \{0,1\}^*$, we define

$$L(x) = \begin{cases} 1, & \text{if } x \in L, \\ 0, & \text{otherwise.} \end{cases}$$

Definition (The classes BPTIME and BPP)

- For $T : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$, we say that a PTM M **decides L in time $T(n)$** if, for every $x \in \{0,1\}^*$:
  - M halts in $T(|x|)$ steps regardless of its random choices;
  - $\Pr[M(x) = L(x)] \geq \frac{2}{3}$.
- We let $\text{BPTIME}(T(n))$ be the class of languages decided by PTMs in $O(T(n))$ time and define

$$\text{BPP} = \bigcup_c \text{BPTIME}(n^c).$$

Possible Modifications

- The PTM in the previous definition satisfies a very strong "excluded middle" property:

  For every input, it either accepts it with probability at least $\frac{2}{3}$ or rejects it with probability at least $\frac{2}{3}$.

- Several modifications are possible without affecting the classes BPTIME($T(n)$) and BPP.
  - The constant $\frac{2}{3}$ can be replaced with any other constant greater than half;
  - We may allow "unfair" coins;
  - We may allow the machine to run in expected polynomial-time.

Worst-Case Character and Relation With P

- The definition allows the PTM M, on input x, to output a value different from $L(x)$, i.e., the wrong answer, with positive probability.
- However, this probability is only over the random choices that M makes in the computation.
- Thus, BPP, like P, is still a class capturing complexity on worst-case inputs.
- A deterministic TM is a special case of a PTM (where both transition functions are equal).
- So the class BPP contains P.

Alternative Definition of BPP

- We define BPP using deterministic TMs where the sequence of "coin tosses" are provided to the TM as an additional input.

Definition (Alternative Definition of BPP)

A language L is in BPP if there exists a polynomial-time TM M and a polynomial $p : \mathbb{N} \to \mathbb{N}$, such that, for every $x \in \{0,1\}^*$,

$$\Pr_{r \in_R \{0,1\}^{p(|x|)}}[M(x,r) = L(x)] \geq \tfrac{2}{3}.$$

- Note that, in time $2^{\text{poly}(n)}$, it is possible to enumerate all the possible random choices of a polynomial-time PTM.
- This makes it clear that $\text{BPP} \subseteq \text{EXP}$.

BPP and P

- Currently, we only know that

$$\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{EXP}.$$

- On the other hand, we are even unable to show that BPP is a proper subset of NEXP.
- A central open question is whether or not $\mathsf{BPP} = \mathsf{P}$.
- Many complexity theorists believe that $\mathsf{BPP} = \mathsf{P}$, that is, that there is a way to transform every probabilistic algorithm to a deterministic algorithm, while incurring only a polynomial slowdown.

Subsection 2

Some Examples of PTMs

Finding a Median

- A **median** of a set of numbers
$$\{a_1, \ldots, a_n\}$$
is any number x, such that:
  - At least $\frac{n}{2}$ of the a_i's are smaller than or equal to x;
  - At least $\frac{n}{2}$ of them are larger that or equal to x.
- One way to find a median of a given set of numbers is to:
  - Sort the numbers;
  - Output the $\frac{n}{2}$ smallest of them.
- This takes $O(n \log n)$ time (assuming that we can perform basic operations on each number at unit cost).

A Probabilistic Algorithm Finding a Median

- We show a simple probabilistic algorithm to find the median in $O(n)$ time.
- There are also known linear time deterministic algorithms for this problem.
- However, the following probabilistic algorithm is still the simplest and most practical known.
- The probabilistic algorithm actually solves a more general problem.
- It finds the k-th smallest number in the set, for every k.

The Algorithm FindKthElement

FindKthElement$(k, a_1, \ldots, a_n)$

1. Pick a random $i \in [n]$ and let $x = a_i$.
2. Scan $\{a_1, \ldots, a_n\}$ and count the number m of a_i's, with $a_i \leq x$.
3. If $m = k$, then output x.
4. Otherwise, if $m > k$, then:
   - Copy to a new list L all elements, such that $a_i \leq x$;
   - Run FindKthElement(k, L).
5. Otherwise (if $m < k$):
   - Copy to a new list H all elements, such that $a_i > x$;
   - Run FindKthElement$(k - m, H)$.

Correctness and Complexity

- It is clear that $\textsc{FindKthElement}(k, a_1, \ldots, a_n)$ outputs the k-th smallest element.
- In the worst case, where $k = \frac{n}{2}$, we expect to get a new list with roughly $\frac{3}{4}n$ elements.
- So we expect that, in each recursive call, the number of elements will shrink by at least $\frac{n}{10}$.
- Thus,
$$T(n) = \mathrm{O}(n) + T\left(\frac{9}{10}n\right).$$
- This implies
$$T(n) = \mathrm{O}(n).$$

Time Complexity of FindKthElement

Claim (Time Complexity of FindKthElement)

For every input $k, a_1, \ldots, a_n$ to FindKthElement, let $T(k, a_1, \ldots, a_n)$ be the expected number of steps the algorithm takes on this input. Let $T(n)$ be the maximum of $T(k, a_1, \ldots, a_n)$ over all length n inputs. Then

$$T(n) = O(n).$$

- All nonrecursive operations can be executed in a linear number of steps cn, for some constant c.

  We show, by induction,

  $$T(n) \leq 10cn.$$

  Fix some input $k, a_1, \ldots, a_n$.

Time Complexity of FindKthElement (Cont'd)

- For every $j \in [n]$, we choose x to be the j-th smallest element of $a_1, \ldots, a_n$ with probability $\frac{1}{n}$, and perform one of the following:
  - At most $T(j)$ steps, if $j > k$;
  - At most $T(n-j)$ steps, if $j < k$.

  As preparation, consider the function

$$f(x) = \frac{n(n+1)}{2} + nx - x^2.$$

  It attains its maximum value at $x = \frac{n}{2}$.

  That maximum value is

$$y_{\max} = \frac{3n^2 + 2n}{4} \overset{\text{large } n}{\leq} \frac{9n^2}{10}.$$

Time Complexity of FindKthElement (Cont'd)

- Putting everything together,

$$\begin{array}{rcl} T(k,a_1,\ldots,a_n) & \leq & cn+\frac{1}{n}(\sum_{j>k}T(j)+\sum_{j<k}T(n-j)) \\ & \leq & cn+\frac{10c}{n}(\sum_{j>k}j+\sum_{j<k}(n-j)) \\ & \leq & cn+\frac{10c}{n}(\sum_{j>k}j+kn-\sum_{j<k}j) \\ & \leq & cn+\frac{10c}{n}\left(\frac{n(n+1)}{2}-\frac{k(k+1)}{2}+kn-\frac{k(k-1)}{2}\right) \\ & = & cn+\frac{10c}{n}\left(\frac{n(n+1)}{2}+kn-k^2\right) \\ & \stackrel{\text{large } n}{\leq} & cn+\frac{10c}{n}\frac{9n^2}{10} \\ & = & 10cn. \end{array}$$

Primality Testing

- In **primality testing**, we are given an integer N and wish to determine whether or not it is prime.
- We want efficient algorithms that run in time polynomial in the size of N's representation, i.e., $\text{poly}(\log N)$ time.
  - In the 1970s efficient probabilistic algorithms for primality testing were discovered.
  - In a very recent breakthrough, Agrawal, Kayal, and Saxena (2004) gave a deterministic polynomial-time algorithm for primality testing.
- Formally, primality testing consists of checking membership in

$$\textsc{Primes} = \{\llcorner N \lrcorner : N \text{ is a prime number}\}.$$

- We sketch an algorithm showing that $\textsc{Primes}$ is in BPP.

Quadratic Residues

- For every number N, and $A \in [N-1]$, define

$$\mathrm{QR}_N(A) = \begin{cases} 0 & \text{if } \gcd(A,N) \neq 1 \\ +1, & \text{if } \begin{array}{l} A \text{ is a } \textbf{quadratic residue} \text{ modulo } N, \\ \text{i.e., } A = B^2 (\text{mod } N), \text{ for a } B, \text{ such} \\ \text{that } \gcd(B,N) = 1 \end{array} \\ -1, & \text{otherwise} \end{cases}$$

- We use some facts from elementary number theory.
- For odd prime N and $A \in [N-1]$,

$$\mathrm{QR}_N(A) = A^{\frac{N-1}{2}} \pmod{N}.$$

Quadratic Residues (Cont'd)

- For every odd N, A, such that
$$N = \prod_{i=1}^{k} P_i,$$
where $P_1, \ldots, P_k$ are all the (not necessarily distinct) prime factors of N, define the **Jacobi symbol**
$$\left(\frac{N}{A}\right) = \prod_{i=1}^{k} \mathrm{QR}_{P_i}(A).$$
- The Jacobi symbol is computable in time $O(\log A \cdot \log N)$.
- For odd composite N, among all $A \in [N-1]$, such that $\gcd(N, A) = 1$, at most half of the A's satisfy
$$\left(\frac{N}{A}\right) = A^{\frac{N-1}{2}} \pmod{N}.$$

Algorithm for Primality Testing

- We assume, without loss of generality, that N is odd.

  The facts above imply a simple algorithm for testing primality of N.
  - Choose a random $1 \leq A < N$.
    - If $\gcd(N, A) > 1$ or $\left(\frac{N}{A}\right) \neq A^{(N-1)/2} \pmod{N}$, then output "composite";
    - Otherwise, output "prime".
- Correctness:
  - This algorithm will always output "prime", if N is prime.
  - If N is composite, it will output "composite" with probability $\geq \frac{1}{2}$. The probability can be amplified by repeating the test a constant number of times.
- The search problem corresponding to primality testing - finding the factorization of a given composite number N - seems very different and much more difficult.

Algebraic Circuits

- We assume given a polynomial in the form of an algebraic circuit.

  This is analogous to the notion of a Boolean circuit, but instead of the operators $\wedge, \vee$ and $\neg$, we have the operators $+, -$ and $\times$.
- Formally, an n-**variable algebraic circuit** is a directed acyclic graph with:
  - The sources labeled by a variable name from the set $x_1, \ldots, x_n$;
  - Each nonsource node having in-degree two is labeled by an operator from the set $\{+, -, \times\}$;
  - A single sink, which we call the **output node**.
- The algebraic circuit defines a polynomial from $\mathbb{Z}^n$ to $\mathbb{Z}$ by:
  - Placing the inputs on the sources;
  - Computing the value of each node using the appropriate operator.
- The circuit computes a function $f(x_1, x_2, \ldots, x_n)$ of the inputs that can be described by a multivariate polynomial in $x_1, x_2, \ldots, x_n$.

Polynomial Identity Testing

- We consider the following problem.
- Given a polynomial with integer coefficients in an algebraic circuit form, decide whether this polynomial is in fact identically zero.
- We define ZeroP to be the set of algebraic circuits that compute the identically zero polynomial.
- Note that we can reduce the problem of deciding whether two circuits C, C' compute the same polynomial to ZeroP by constructing the circuit D such that

$$D(x_1, \ldots, x_n) = C(x_1, \ldots, x_n) - C'(x_1, \ldots, x_n).$$

- So determining membership in ZeroP is also called **polynomial identity testing**.

Compactness of Represenation

- The ZeroP problem is nontrivial.
- The reason is that a very compact circuit can represent polynomials with a large number of terms.

  Example: Consider the polynomial

  $$\prod_i (1 + x_i).$$

  It can be computed using a circuit of size $2n$.

  However, it has 2^n terms if we open all parentheses.

The Schwartz-Zippel Lemma

- There is a simple and efficient probabilistic algorithm for testing membership in $\textsc{ZeroP}$, using the Schwartz-Zippel Lemma.

Lemma (Schwartz-Zippel Lemma)

Let

$$p(x_1, x_2, \ldots, x_m)$$

be a nonzero polynomial of total degree at most d.
Let S be a finite set of integers.
Then, if $a_1, a_2, \ldots, a_m$ are randomly chosen with replacement from S,

$$\Pr[p(a_1, a_2, \ldots, a_m) \neq 0] \geq 1 - \frac{d}{|S|}.$$

A Probabilistic Algorithm for ZeroP

- A size m circuit C contains at most m multiplications.
- So, it defines a polynomial of degree at most 2^m.
- This suggests the simple probabilistic algorithm:
    - Choose n numbers $x_1, \ldots, x_n$ from 1 to $10 \cdot 2^m$.
      This requires $O(n \cdot m)$ random bits.
    - Evaluate the circuit C on $x_1, \ldots, x_n$ to obtain an output y.
    - Accept if $y = 0$.
    - Reject, otherwise.
- Clearly if $C \in \mathrm{ZeroP}$, then we always accept.
- By the lemma, if $C \notin \mathrm{ZeroP}$, then we will reject with probability at least $\frac{9}{10}$.

Using Fingerprinting

- There is a problem with the algorithm.

  The degree of the polynomial represented by the circuit can be as high as 2^m.

  So the output y and other intermediate values arising in the computation may be as large as $(10 \cdot 2^m)^{2^m}$.

  Such values require exponentially many bits just to write down!

- We solve the problem using **fingerprinting**.

  The idea is to perform the evaluation of C on $x_1, \ldots, x_n$ modulo a number k that is chosen at random in $[2^{2m}]$.

- Instead of $y = C(x_1, \ldots, x_n)$, we compute the value $y \pmod{k}$.
  - If $y = 0$, then $y \pmod{k}$ is also equal to 0.
  - If $y \neq 0$, then we show that with probability at least $\delta = \frac{1}{4m}$, k does not divide y.

- This suffices because we can repeat this procedure $\mathrm{O}\left(\frac{1}{\delta}\right)$ times and accept only if the output is zero in all these repetitions.

Proof of the Claim

Claim

If $y \neq 0$, then, with probability at least $\delta = \frac{1}{4m}$, k does not divide y.

- Assume that $y \neq 0$.

  Let $\mathcal{B} = \{p_1, \ldots, p_\ell\}$ denote the set of distinct prime factors of y.

  It is sufficient to show that, with probability at least δ, the number k will be a prime number not in $\mathcal{B}$.

  By the Prime Number Theorem, for sufficiently large m, the number of primes in $[2^{2m}]$ is at least $\frac{2^{2m}}{2m}$.

  Now y can have at most $\log y \leq 5m \cdot 2^m = o\left(\frac{2^{2m}}{2m}\right)$ prime factors.

  So, for sufficiently large m, the number of k's in $[2^{2m}]$, such that k is prime and is not in $\mathcal{B}$ is at least $\frac{2^{2m}}{4m}$.

  Therefore, a random k has this property with probability $\geq \frac{1}{4m} = \delta$.

Perfect Matchings in Bipartite Graphs

- Let $G = (V, E)$ be a **bipartite graph** with two equal parts.
- That is, $V = V_1 \cup V_2$, where:
  - V_1, V_2 are disjoint and of equal size;
  - $E \subseteq V_1 \times V_2$.
- A **perfect matching** in G is a subset of edges $E' \subseteq E$, such that every vertex appears exactly once in E'.
- Set $n = |V_1| = |V_2|$.
- Identify both V_1 and V_2 with the set $[n]$.
- We may think of E' as a permutation $\sigma : [n] \to [n]$ mapping every $i \in [n]$ to the unique $j \in [n]$, such that $\overline{ij} \in E'$.
- Several deterministic algorithms are known for detecting if a perfect matching exists in a given graph.
- We describe a simple randomized algorithm, due to Lovász, using the Schwartz-Zippel Lemma.

Matrix Associated with a Bipartite Graph

- Let $G = (V, E)$ be a $2n$-vertex bipartite graph.
- Let X be an $n \times n$ matrix of real variables whose (i, j)-th entry $X_{i,j}$ is given by

$$X_{i,j} = \begin{cases} x_{i,j}, & \text{if } \overline{ij} \in E, \\ 0, & \text{otherwise,} \end{cases}$$

  where $x_{i,j}$ is a variable.
- The **determinant** of a matrix A is defined by

$$\det(A) = \sum_{\sigma \in S_n} (-1)^{\text{sgn}(\sigma)} \prod_{i=1}^{n} A_{i,\sigma(i)},$$

  where:
  - S_n is the set of all permutations of $[n]$;
  - $\text{sgn}(\sigma)$ is the parity of the number of transposed pairs in σ. That is, pairs $\langle i, j \rangle$, such that $i < j$, but $\sigma(i) > \sigma(j)$.

Matrix Associated with a Bipartite Graph (Cont'd)

- $\det(X)$ is a degree n polynomial in the variables $\{x_{i,j}\}_{\overline{ij} \in E}$.
- Moreover, it has a monomial for every perfect matching that exists in the graph.
- Thus, G has a perfect matching if and only if $\det(X)$ is not the identically zero polynomial.
- $\det(X)$ may have exponentially many monomials.
- However, for every setting of values to the $x_{i,j}$ variables, $\det(X)$ can be efficiently evaluated using the well-known algorithm for computing determinants.

Testing for Perfect Matching in a Bipartite Graph

- The preceding observations lead to Lovász's randomized algorithm.
  - Pick random values for $x_{i,j}$'s from $[2n]$.
  - Substitute them in X, and compute the determinant.
  - If the determinant is nonzero, output "accept".
  - Else output "reject".

Subsection 3

One-Sided and "Zero-Sided" Error: RP, coRP, ZPP

Two-Sided versus One-Sided Error

- The class BPP captures what we call probabilistic algorithms with two-sided error.
- This allows an algorithm for a language L to output (with some small probability) both:
  - 0, when $x \in L$;
  - 1, when $x \notin L$.
- Many probabilistic algorithms have the property of one-sided error.
- For instance:
  - If $x \notin L$, they never output 1;
  - If $x \in L$, they may output 0.

The Class RP

Definition (The Class RP)

The class RTIME($T(n)$) contains every language L for which there is a probabilistic TM M running in $T(n)$ time, such that

$$x \in L \quad \Rightarrow \quad \Pr[M(x) = 1] \geq \frac{2}{3};$$
$$x \notin L \quad \Rightarrow \quad \Pr[M(x) = 0] = 1.$$

We define

$$\mathrm{RP} = \bigcup_{c>0} \mathrm{RTIME}(n^c).$$

RP, NP, BPP, coRP

- We have
$$\mathrm{RP} \subseteq \mathrm{NP}.$$
- In fact, every accepting branch is a "certificate" of membership.
- Note that it is not known if
$$\mathrm{BPP} \subseteq \mathrm{NP}.$$
- The class
$$\mathrm{coRP} = \{\overline{L} : L \in \mathrm{RP}\}$$
captures one-sided error algorithms that:
  - May output 1, when $x \notin L$;
  - Never output 0, if $x \in L$.

Expected Running Time

- For a PTM M, and input x, we define the random variable $T_{M,x}$ to be the running time of M on input x.
- Then we have
$$\Pr[T_{M,x} = T] = p,$$
if, with probability p over the random choices of M on input x, M halts within T steps.
- We say that M has **expected running time** $T(n)$ if, for all $x \in \{0,1\}^*$, the expectation
$$E[T_{M,x}] \leq T(|x|).$$

"Zero-Sided" Error

- We now define PTMs that never err.
- They are called "zero error" machines.

Definition (The class ZTIME($T(n)$))

The class ZTIME($T(n)$) contains all the languages L for which there is a machine M that runs in an expected-time $O(T(n))$, such that, for every input x, whenever M halts on x, its output $M(x)$ is exactly $L(x)$.
We define

$$\mathrm{ZPP} = \bigcup_{c>0} \mathrm{ZTIME}(n^c).$$

Relations Between Probabilistic Classes

- The question whether or not $\mathsf{P} = \mathsf{NP} \cap \mathsf{coNP}$ is open.
- But for probabilistic classes we have:

Theorem

$$\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}.$$

- We skip the proof.
- The following relations hold between probabilistic complexity classes:

$$\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}, \quad \mathsf{RP} \subseteq \mathsf{BPP}, \quad \mathsf{coRP} \subseteq \mathsf{BPP}.$$

Subsection 4

The Robustness of Our Definitions

Role of Precise Constants

- In the definition of BPP, the choice of the constant $\frac{2}{3}$ is arbitrary.
- We show that we can replace $\frac{2}{3}$ with any constant larger than $\frac{1}{2}$ and, in fact, even with $\frac{1}{2} + n^{-c}$ for a constant $c > 0$.

Lemma

For $c > 0$, let $\text{BPP}_{\frac{1}{2}+n^{-c}}$ denote the class of languages L for which there is a polynomial-time PTM M, satisfying

$$\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c},$$

for every $x \in \{0,1\}^*$. Then

$$\text{BPP}_{\frac{1}{2}+n^{-c}} = \text{BPP}.$$

Strategy

- Clearly, $\mathrm{BPP} \subseteq \mathrm{BPP}_{1/2+n^{-c}}$.
- So to prove the lemma we need to show that we can transform a machine with success probability $\frac{1}{2} + n^{-c}$ into a machine with success probability $\frac{2}{3}$.
- We do much better by transforming such a machine into a machine with success probability exponentially close to one!

Error Reduction for BPP

Theorem (Error Reduction for BPP)

Let $L \in \{0,1\}^*$ be a language. Suppose there exists a polynomial-time PTM M, such that, for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}.$$

Then, for every constant $d > 0$, there exists a polynomial-time PTM M', such that, for every $x \in \{0,1\}^*$,

$$\Pr[M'(x) = L(x)] \geq 1 - 2^{-|x|^d}.$$

Error Reduction for BPP (Cont'd)

- The machine M' simply does the following:
  - For every input $x \in \{0,1\}^*$, run $M(x)$, for $k = 8|x|^{2c+d}$ times obtaining k outputs $y_1, \ldots, y_k \in \{0,1\}^*$.
  - If the majority of these outputs is 1, then output 1.
  - Otherwise, output 0.

For $i \in [k]$, define the random variable

$$X_i = \begin{cases} 1, & \text{if } y_i = L(x), \\ 0, & \text{otherwise.} \end{cases}$$

The random variables $X_1, \ldots, X_k$ are independent.

Moreover, for $p = \frac{1}{2} + |x|^{-c}$,

$$E[X_i] = \Pr[X_i = 1] \geq p.$$

Error Reduction for BPP (Cont'd)

- By the Chernoff bound, for sufficiently small δ,

$$\Pr\left[\left|\sum_{i=1}^{k} X_i - pk\right| > \delta pk\right] < e^{-\frac{\delta^2}{4}pk}.$$

Let

$$p = \frac{1}{2} + |x|^{-c} \quad \text{and} \quad \delta = \frac{1}{2}|x|^{-c}.$$

For those p and δ, if

$$\sum_{i=1}^{k} X_i \geq pk - \delta pk,$$

then M' outputs the right answer.

So the probability M' outputs a wrong answer is

$$\leq e^{-\frac{1}{16|x|^{2c}}(\frac{1}{2}+|x|^{-c})8|x|^{2c+d}} \leq e^{-\frac{1}{4|x|^{2c}}\frac{1}{2}8|x|^{2c+d}} \leq 2^{-|x|^d}.$$

Remarks on Error Reduction

- A similar result holds for the one-sided error classes RP and coRP.
- In that case, we can even change $\frac{2}{3}$ to values smaller than $\frac{1}{2}$.
- Thus, we can take a probabilistic algorithm that succeeds with quite modest probability and transform it into an algorithm that succeeds with overwhelming probability.

  Even for moderate values of n, an error probability that is of the order of 2^{-n} is so small that, for all practical purposes, probabilistic algorithms are just as good as deterministic algorithms.
- The proof uses $O(k)$ independent repetitions to transform an algorithm with success probability $\frac{2}{3}$ into an algorithm with success probability $1 - 2^{-k}$.

  Thus, if the original used m random coins, then the new algorithm will use $O(km)$ coins.

  Surprisingly, there is a transformation that only uses $O(m + k)$ random coins to achieve the same error reduction.

Expected Versus Worst-Case Running Time

- When defining RTIME($T(n)$) and BPTIME($T(n)$), we required the machine to halt in $T(n)$ time regardless of its random choices.
- We could have used expected running time instead, as in the definition of ZPP.
- It turns out this yields an equivalent definition.
- We can transform a PTM M whose expected running time is $T(n)$ to a PTM M' that always halts after at most $100T(n)$ steps by:
  - Adding a counter;
  - Halting with an arbitrary output after too many steps have gone by.
- By Markov's inequality, the probability that M runs for more than $100T(n)$ steps is at most $\frac{1}{100}$.
- So the transformation changes the acceptance probability by at most $\frac{1}{100}$.

More General Random Choices Than a Fair Coin

- One could conceive of real-life computers that have a "coin" that comes up heads with probability ρ that is not $\frac{1}{2}$, called a ρ-**coin**.
- It turns out, such a coin will not give probabilistic algorithms new power, if ρ is efficiently computable.

Lemma

A coin with

$$\Pr[\text{Heads}] = \rho$$

can be simulated by a PTM in expected time $O(1)$, provided the i-th bit of ρ is computable in $\text{poly}(i)$ time.

- Let the binary expansion of ρ be

$$\rho = 0.p_1 p_2 p_3 \ldots.$$

More General Random Choices (Cont'd)

- The simulating PTM:
  - Generates a sequence of random bits

$$b_1, b_2, \ldots,$$

    one by one, where b_i is generated at step i.
  - If $b_i < p_i$, then outputs "heads" and stops.
  - If $b_i > p_i$, then outputs "tails" and halts.
  - Otherwise, it goes to step $i + 1$.

The machine reaches step $i + 1$ iff $b_j = p_j$, for all $j \leq i$.

This happens with probability $\frac{1}{2^i}$.

Thus, the probability of "heads" is $\sum_i p_i \frac{1}{2^i}$.

This is exactly ρ.

Furthermore, the expected running time is $\sum_i i^c \cdot \frac{1}{2^i}$.

For every constant c, this sum is bounded by another constant.

Simulating Fair via Available Biased Coins

- Probabilistic algorithms that only have access to ρ-coins do not have less power than standard probabilistic algorithms.

Lemma (von-Neumann)

A coin with $\Pr[\text{Heads}] = \frac{1}{2}$ can be simulated by a probabilistic TM with access to a stream of ρ-biased coins in expected time $O\left(\frac{1}{\rho(1-\rho)}\right)$.

- The following TM M, given the ability to toss ρ-coins, outputs a $\frac{1}{2}$-coin.
  - M tosses pairs of coins until the first time it gets a pair containing two different results (i.e., "Heads-Tails" or "Tails-Heads").
  - If the first of these two is "Heads", then outputs "Heads".
  - Otherwise, it outputs "Tails".

Simulating Fair via Available Biased Coins (Cont'd)

- The probability of getting "Head-Tails" is $\rho(1-\rho)$.

  The probability of "Tails-Heads" is $(1-\rho)\rho = \rho(1-\rho)$.

  Hence, in each step, M halts with probability $2\rho(1-\rho)$.

  Conditioned on M halting in a particular step, the outputs "Heads" and "Tails" are equiprobable.

  Equivalently, M's output is a fair coin.

  Knowing ρ is not needed to run this simulation.

Subsection 5

Relationship Between BPP and Other Classes

Summary of Results

- $\mathsf{BPP} \subseteq \mathsf{P}_{/\mathsf{poly}}$.

  It follows $\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{P}_{/\mathsf{poly}}$.
- Furthermore, $\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$.

  So, if $\mathsf{NP} = \mathsf{P}$, then $\mathsf{BPP} = \mathsf{P}$.

  Since we do not believe $\mathsf{P} = \mathsf{NP}$, this still leaves open the possibility that $\mathsf{P} \neq \mathsf{BPP}$.
- We can show that, if certain plausible complexity-theoretic conjectures are true, then $\mathsf{BPP} = \mathsf{P}$.

  Thus, we suspect that BPP is the same as P.

  Hence (by the Time Hierarchy Theorem) BPP is a proper subset of, say, $\mathsf{DTIME}(n^{\log n})$.

  Yet, currently, we are not even able to show that BPP is a proper subset of NEXP.

$\mathsf{BPP} \subseteq \mathsf{P}_{/\mathsf{poly}}$

- We show that all BPP languages have polynomial sized circuits.
- By the Karp-Lipton Theorem, unless the polynomial hierarchy collapses, $3\textsc{Sat}$ cannot be solved in probabilistic polynomial time.

Theorem

$$\mathsf{BPP} \subseteq \mathsf{P}_{/\mathsf{poly}}.$$

- Suppose $L \in \mathsf{BPP}$.

  By the alternative definition of BPP and the error reduction, there exists a TM M that on inputs of size n uses m random bits, such that, for $x \in \{0,1\}^n$,

$$\Pr_r[M(x,r) \neq L(x)] \leq 2^{-n-1}.$$

$\mathsf{BPP} \subseteq \mathsf{P}_{/\mathrm{poly}}$ (Cont'd)

- Say that a string $r \in \{0,1\}^m$ is **bad** for an input $x \in \{0,1\}^n$ if
$$M(x,r) \neq L(x).$$
Otherwise, call r **good** for x.
For every x, at most $\frac{2^m}{2^{n+1}}$ strings r are bad for x.
Adding over all $x \in \{0,1\}^n$, at most
$$2^n \cdot \frac{2^m}{2^{n+1}} = \frac{2^m}{2}$$
strings r are bad for some x.
So, there exists $r_0 \in \{0,1\}^m$ that is good for every $x \in \{0,1\}^n$.
We hardwire such a string r_0.
We obtain a circuit C (of size at most quadratic in the running time of M) that, on input x, outputs $M(x,r_0)$.
C satisfies
$$C(x) = L(x), \quad \text{for every } x \in \{0,1\}^n.$$

BPP is in PH

Theorem (Sipser-Gács Theorem)

$$\mathsf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p.$$

- Note BPP is closed under complementation, i.e., BPP = coBPP.

  So it suffices to prove that $\mathsf{BPP} \subseteq \Sigma_2^p$.

  Suppose $L \in \mathsf{BPP}$.

  By error reduction, there exists a polynomial-time deterministic TM M for L that on inputs of length n uses $m = \text{poly}(n)$ random bits, such that:
  - $x \in L$ implies $\Pr_r[M(x,r) \text{ accepts}] \geq 1 - 2^{-n}$;
  - $x \notin L$ implies $\Pr_r[M(x,r) \text{ accepts}] \leq 2^{-n}$.

The Sipser-Gács Theorem (Cont'd)

- For $x \in \{0,1\}^n$, let

$$S_x = \{r : M \text{ accepts } \langle x, r \rangle\}.$$

Then:

- $|S_x| \geq (1 - 2^{-n})2^m$, if $x \in L$;
- $|S_x| \leq 2^{-n}2^m$, if $x \notin L$.

We show how to check, using two quantifiers, which of these is true.

For a set $S \subseteq \{0,1\}^m$ and string $u \in \{0,1\}^m$, we denote by $S + u$ the "shift" of the set S by u,

$$S + u = \{x + u : x \in S\},$$

where $+$ denotes vector addition modulo 2, i.e., bitwise XOR.

The Sipser-Gács Theorem (Claim 1)

- Let
$$k = \left\lceil \frac{m}{n} \right\rceil + 1.$$
Claim: For every set $S \subseteq \{0,1\}^m$, with $|S| \leq 2^{m-n}$, and every k vectors $u_1, \ldots, u_k$,
$$\bigcup_{i=1}^{k} (S + u_i) \neq \{0,1\}^m.$$
We have $|S + u_i| = |S|$.
So, by the union bound, for sufficiently large n,
$$\left| \bigcup_{i=1}^{k} (S + u_i) \right| \leq k|S| < 2^m.$$

The Sipser-Gács Theorem (Claim 2)

Claim: For every set $S \subseteq \{0,1\}^m$, with $|S| \geq (1-2^{-n})2^m$, there exist $u_1,\ldots,u_k$, such that

$$\bigcup_{i=1}^{k}(S+u_i)=\{0,1\}^m.$$

Suppose $u_1,\ldots,u_k$ are chosen independently at random.
We show that, then,

$$\Pr\left[\bigcup_{i=1}^{k}(S+u_i)=\{0,1\}^m\right]>0.$$

Indeed, for $r\in\{0,1\}^m$, let B_r denote the "bad event" that

$$r\notin\bigcup_{i=1}^{k}(S+u_i).$$

The Sipser-Gács Theorem (Claim 2 Cont'd)

- It suffices to prove that

$$\Pr[\exists r \in \{0,1\}^m B_r] < 1.$$

This will follow by the union bound if we can show, for every r, that

$$\Pr[B_r] < 2^{-m}.$$

Consider the events

$$B_r^i = \{r \notin S + u_i\} = \{r + u_i \notin S\}$$

(the second equality because mod 2, $a + b = c$ iff $a = c + b$).

The Sipser-Gács Theorem (Claim 2 Cont'd)

- We have
$$B_r = \bigcap_{i \in [k]} B_r^i.$$
Now $r + u_i$ is a uniform element in $\{0,1\}^m$.
So it will be in S with probability at least $1 - 2^{-n}$.
But the B_r^i are independent for different i's.
So
$$\Pr[B_r] = \Pr[B_r^i]^k \leq 2^{-nk} < 2^{-m}.$$

The Sipser-Gács Theorem (Cont'd)

- The claims show that $x \in L$ if and only if the following statement is true:

$$\exists u_1, \ldots, u_k \in \{0,1\}^m \forall r \in \{0,1\}^m \; r \in \bigcup_{i=1}^{k} (S_x + u_i).$$

Equivalently,

$$\exists u_1, \ldots, u_k \in \{0,1\}^m \forall r \in \{0,1\}^m \; \bigvee_{i=1}^{k} M(x, r \oplus u_i) \text{ accepts.}$$

This represents a Σ_2^p computation since k is poly(n).

Hence, we have shown $L \in \Sigma_2$.

Complete Problems for BPP?

- We know of no complete languages for BPP.
- One reason is that the defining property of BPTIME machines is semantic.
- They accept every input string either with probability at least $\frac{2}{3}$ or with probability at most $\frac{1}{3}$.
- Testing whether a given TM M has this property is undecidable.
- By contrast, the defining property of an NDTM is syntactic.
- Given a string, it is easy to determine if it is a valid encoding of an NDTM.

Complete Problems for BPP? (Cont'd)

- Consider the following natural attempt at a BPP-complete language.
- Define L to contain all tuples $\langle M, x, 1^t \rangle$, such that, on input x, M outputs 1 within t steps with probability at least $\frac{2}{3}$,

$$L = \{\langle M, x, 1^t \rangle : \text{on input } x, M \text{ outputs 1 within } t \text{ steps, with probability at least } \tfrac{2}{3}\}.$$

- The language L is indeed BPP-hard.
- However, it is not known to be in BPP.
- For $\langle M, x, 1^t \rangle \in L$ we could have $\Pr[M(x) = 1] = \frac{1}{2}$, say, which is greater than $\frac{1}{3}$.
- We note that this language is #P-complete.
- Hence, it is unlikely to be in any level of the polynomial hierarchy unless the hierarchy collapses.
- But, if BPP = P, then BPP has a complete problem (since P does).

Does BPTIME Have a Hierarchy Theorem?

- Is every problem in BPTIME(n^2) also in BPTIME(n)?
- One would imagine not.
- This seems like the kind of result we should be able to prove using the diagonalization techniques.
- However, currently we are even unable to show that, say,

$$\text{BPTIME}(n) \neq \text{BPTIME}(n^{(\log n)^{10}}).$$

- The standard diagonalization techniques fail, again apparently because the defining property of BPTIME machines is semantic.

Subsection 6

Randomized Reductions

Randomized Polynomial Time Reduction

- For randomized algorithms, it makes sense to define a notion of randomized reduction between two languages.

Definition (Randomized Polynomial Time Reduction)

Language B **reduces to** language C **under a randomized polynomial time reduction**, denoted $B \leq_r C$, if there is a probabilistic TM M, such that, for every $x \in \{0,1\}^*$,

$$\Pr[B(x) = C(M(x))] \geq \frac{2}{3}.$$

- This notion of reduction is not transitive.
- Nevertheless, it is useful because

$$C \in \text{BPP} \quad \text{and} \quad B \leq_r C \quad \text{imply} \quad B \in \text{BPP}.$$

The Class BP · NP

- Arguably BPP is as good as P as a formalization of the notion of efficient computation.
- So one could conceivably define NP-completeness using randomized reductions instead of deterministic reductions.
- The Cook-Levin Theorem shows that NP may be defined as the set

$$\mathsf{NP} = \{L : L \leq_p 3\textsc{Sat}\}.$$

- If we replace "deterministic polynomial-time reduction" with "randomized reduction", then we obtain a somewhat different class.

Definition (The Class BP · NP)

We define the class

$$\mathsf{BP} \cdot \mathsf{NP} = \{L : L \leq_r 3\textsc{Sat}\}.$$

A Property of BP · NP

- One interesting application is given by a randomized reduction (encountered later) from $3\textsc{Sat}$ to solving a special case of $3\textsc{Sat}$, where we are guaranteed that the formula is either unsatisfiable or has a single unique satisfying assignment.
- As far as class containments, we have the following

Proposition

If $\overline{3\textsc{Sat}} \in \mathsf{BP} \cdot \mathsf{NP}$, then PH collapses to Σ_3^p.

- The Proposition provides evidence that $\overline{3\textsc{Sat}} \leq_r 3\textsc{Sat}$ is unlikely.

Subsection 7

Randomized Space-Bounded Computation

The classes BPL and RL

- A PTM **uses space** $S(n)$ if, in any branch of its computation on a length n input, the number of work-tape cells that are ever nonblank is at most $\mathrm{O}(S(n))$.
- The classes BPL and RL are the two-sided error and one-sided error probabilistic analogs of the class L.

Definition (The classes BPL and RL)

- A language L is in BPL if there is a $\mathrm{O}(\log n)$-space probabilistic TM M, such that
$$\Pr[M(x) = L(x)] \geq \frac{2}{3}.$$
- A language L is in RL if there is a $\mathrm{O}(\log n)$-space probabilistic TM M, such that:
  - If $x \in L$, then $\Pr[M(x) = 1] \geq \frac{2}{3}$;
  - If $x \notin L$, then $\Pr[M(x) = 1] = 0$.

Some Containments

- The error reduction procedure can be implemented with only logarithmic space overhead.
- Therefore, the choice of the precise constant in the preceding definitions is not significant.
- We note that $\mathsf{RL} \subseteq \mathsf{NL}$, whence

$$\mathsf{RL} \subseteq \mathsf{P}.$$

- We also have

$$\mathsf{BPL} \subseteq \mathsf{P}.$$

The Undirected Path Problem

- One famous RL-algorithm is the algorithm for solving UPATH.
- This is the restriction of the NL-complete PATH problem to undirected graphs.

  Given an n-vertex undirected graph G and two vertices s and t, determine whether s is connected to t in G.

Theorem

UPATH $\in$ RL.

- Take a random walk of length $\ell = 100n^4$ starting from s.
  - Initialize the variable v to the vertex s.
  - In each step choose a random neighbor u of v, and set $v \leftarrow u$.
  - Accept iff the walk reaches t within ℓ steps.

Space Complexity and Correctness

- Space Complexity: This is a logspace algorithm.

  It only needs to store:
  - A counter;
  - The index of the current vertex;
  - Some scratch space to compute the next neighbor in the walk.
- Correctness: We have the following cases.

  If s is not connected to t, then the algorithm will never accept.

  It can be shown that if s is connected to t, then the expected number of steps it takes for a walk from s to hit t is at most $10n^4$.

  Hence, our algorithm will accept with probability at least $\frac{3}{4}$.

Comments

- There exists a recent deterministic logspace algorithm for UPATH.
- It is known that

$$\mathsf{RL} \subseteq \mathsf{BPL} \subseteq \mathsf{SPACE}(\log^{3/2} n).$$