Randomized Computation

Matoula Petrolia

December 10, 2009

Randomized Algorithms

Randomized Algorithms: algorithms that can "flip a coin".

Many computational problems seem to be more easily solvable by randomized algorithms.

Some examples:

1. Symbolic Determinants
2. Random Walks
3. The Fermat Test

Symbolic Determinants

- We may need to find out whether the determinant of a matrix A is identically zero or not.
  (For example in the bipartite matching problem, consider the matrix A^G of a graph G - where $a_{ij} = x_{ij}$ if there is an edge connecting nodes i, j and $a_{ij} = 0$ otherwise. Then G has a perfect matching iff $detA \neq 0$.)
- We can calculate the determinant of a matrix using *Gaussian elimination* in polynomial time.
- Using the same method to calculate the determinant of a symbolic matrix can be much harder.

A simple example of a symbolic Gaussian elimination:

$$\begin{pmatrix} x & w & z \\ z & x & w \\ y & z & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} x & w & z \\ 0 & \frac{x^2-zw}{x} & \frac{wx-z^2}{x} \\ 0 & \frac{zx-wy}{x} & -\frac{zy}{x} \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} x & w & z \\ 0 & \frac{x^2-zw}{x} & \frac{wx-z^2}{x} \\ 0 & 0 & -\frac{yz(xz-xw)+(zx-wy)(wx-z^2)}{x(x^2-zw)} \end{pmatrix}$$

Idea: Substitute arbitrary integers for the variables.

- We now get a numerical matrix whose determinant we can calculate in polynomial time.
- If this determinant is not zero then we know that the symbolic determinant is not identically zero.
- *But* we may be unlucky and choose the wrong numbers. That is, the numerical determinant may be zero although the symbolic one is not.

The following Lemma reassures us that choosing the "wrong numbers" is a very unlikely event.

Lemma 1

Let $\pi(x_1, x_2, \ldots, x_m)$ be a polynomial, not identically zero, in m variables each of degree at most d in it and let $M > 0$ be an integer. Then the number of m-tuples $(x_1, x_2, \ldots, x_m) \in \{0, 1, \ldots, M-1\}^m$ such that $\pi(x_1, x_2, \ldots, x_m) = 0$ is at most mdM^{m-1}.

Proof:

By reduction on the number of variables, m. □

The above lemma suggests a randomized algorithm for deciding if a graph G has a perfect matching, that is, if the determinant of the matrix A^G is identically zero.

- $A^G(x_1, \ldots, x_m) = A^G$ with its m variables.
- $det(A^G(x_1, \ldots, x_m))$ is a polynomial in m variables each of degree at most 1 in it.

```
Choose m random integers i_1, i_2, ..., i_m between 0 and
M = 2m.
Compute detA^G(x_1, ..., x_m)) by Gaussian elimination.
If detA^G(x_1, ..., x_m)) ≠ 0, reply "G has a perfect
matching".
If detA^G(x_1, ..., x_m)) = 0, reply "G probably has no
perfect matching".
```

- If the algorithm finds that a matching exists, then we know it does.
- If the algorithm answers "G *probably* has no perfect matching", then there is a possibility of a false negative. The probability of a false negative is no more than $\frac{1}{2}$.

Such algorithms, that can have false negatives with a bounded probability, but no false positives, are called **Monte Carlo** algorithms.

If we perform many independent experiments we can reduce the chance of false negatives: if we repeat k times, then our confidence on the outcome that there is no perfect matching for G increases to $1 - (\frac{1}{2})^k$.

Random Walks

This is a randomized algorithm for SAT:

```
Start with any truth assignment T, and repeat the
following r times:
    If there is no unsatisfied clause, then reply
    "formula is satisfiable" and halt.
    Otherwise, take any unsatisfied clause; pick any
    of its literals at random and flip it, updating
    T.
After r repetitions reply "formula is probably
unsatisfiable".
```

This is *the random walk* algorithm.

Again, for this algorithm we have that

- if the algorithm finds that the formula is satisfiable, then we know it is (no false positives) and
- if the algorithm finds that the formula is *probably* unsatisfiable, then there is a possibility of a false negative.

In other words

- if the formula is unsatisfiable then our algorithm will give us a correct answer but
- if the formula is satisfiable then there is a possibility that a satisfying truth assignment won't be discovered.

The random walk algorithm may perform badly for even a simple satisfiable instance of 3SAT.
But when it comes to 2SAT, the algorithm performs quite decently:

Theorem 1

If the random walk algorithm is applied to a satisfiable instance of 2SAT with n variables, for $r = 2n^2$, then the probability that a satisfying truth assignment will be discoverd is at least $\frac{1}{2}$.

This Theorem implies that the random walk algorithm with $r = 2n^2$ is a *Monte Carlo* algorithm for 2SAT.

To prove this consider:

- $\hat{T}$ is a truth assignment that satisfies the given instance of 2SAT.
- The starting assignment T differs from $\hat{T}$ in i values.
- $t(i)$ is the expected number of repetitions of the flipping step until a satisfying truth assignment is discovered.

Notice that every time the algorithm flips a randomly chosen literal, we have at least $\frac{1}{2}$ chance of moving closer to $\hat{T}$.

Properties of $t(i)$:

- $t(0) = 0$
- $t(n) \leq t(n-1) + 1$
- $t(i) \leq \frac{1}{2}(t(i-1)+1) + \frac{1}{2}(t(i+1)+1)$

We will prove that $t(i) \leq n^2$ and we will use the following Lemma to complete the proof:

Lemma 2

If x is a random variable taking nonnegative integer values, $\mathcal{E}(x)$ is the expected value of x, then $\forall k > 0$ $P[x \geq k \cdot \mathcal{E}(x)] \leq \frac{1}{k}$.

Proof:

Let $p_i = P[x = i]$. Then,

$$\mathcal{E}(x) = \sum_i ip_i = \sum_{i \leq k \cdot \mathcal{E}(x)} ip_i + \sum_{i > k \cdot \mathcal{E}(x)} ip_i > k \cdot \mathcal{E}(x) \cdot P[x > k \cdot \mathcal{E}(x)].$$

$\square$

Now we are ready to prove the Theorem:

Theorem's Proof:

Define $x(i)$ with the following properties:

- $x(0) = 0$
- $x(n) = x(n-1)$
- $x(i) = \frac{1}{2}(x(i-1)+1) + \frac{1}{2}(x(i+1)+1)$

Obviously, $x(i) \geq t(i)$.

Adding together all $x(i)$'s we get $x(1) = 2n-1$ and continuing like this we get $x(i) = 2in - i^2$. When $i = n$, $x(n) = n^2$.

Thus, the expected number of repetitions needed to find a satisfying truth assignment is $t(i) \leq x(i) \leq x(n) = n^2$. Now using Lemma 2, with $k = 2$, we complete the proof. $\square$

The Fermat Test

Theorem 2 (Fermat's Theorem)

If N is a prime number, then for all $0 < a < N$, $a^{N-1} = 1 \bmod N$.

- If $a^{N-1} \neq 1 \bmod N$ then we know that N is composite.
- If $a^{N-1} = 1 \bmod N$ then we can't tell if N is a prime.

Consider the algorithm suggested by Fermat's Theorem:

> `Pick a random residue` a `modulo` N.
> `If` $a^{N-1} \neq 1 \bmod N$ `answer "`N `is composite".`
> `If` $a^{N-1} = 1 \bmod N$ `answer "`N `is probably prime".`

This algorithm would be a polynomial Monte Carlo algorithm if it was true that *for any N not prime $a^{N-1} \neq 1 \bmod N$ for at least half of its nonzero residues.*

But this hypothesis is false: there are some numbers for which $a^{N-1} = 1 \bmod N$ for all of their residues a but still these numbers are composite (*Carmichael* numbers).
(e.g. 561 is not a prime number but still all residues in $\Phi(561)$ pass the Fermat test.)

- The Fermat's test gives a false answer for a Carmichael number.
- There are infinitely many Carmichael numbers.

To overcome this difficulty, we will need the followings:

Definition 1

Let p be an odd prime and $a \neq 0 \bmod p$. Then we define the **Legendre Symbol** $(a|p)$*:*

$$(a|p) = \begin{cases} +1, & a \textit{ is a quadratic residue modulo } p \\ -1, & a \textit{ is not a quadratic residue modulo } p \end{cases}$$

Properties:

- $(a|p) = a^{\frac{p-1}{2}} \bmod p$.
- $(a|p)(b|p) = (ab|p)$.
- $(p|q)(q|p) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}$, p, q odd primes (*Legendre's Law of Quadratic Reciprocity*).

Definition 2

Let M, N be not prime integers and $N = \prod_{i=1}^{n} q_i$ where the q_i's are all odd primes. We define the **Jacobi Symbol** *to be $(M|N) = \prod_{i=1}^{n}(M|q_i)$.*

Properties:

- $(M_1 M_2|N) = (M_1|N)(M_2|N)$.
- $(M + N|N) = (M|N)$.
- If both M, N are odd then $(M|N)(N|M) = (-1)^{\frac{M-1}{2}\frac{N-1}{2}}$.
- $(2|M) = (-1)^{\frac{M^2-1}{8}}$.

Lemma 3

If $\lceil \log MN \rceil = l$, then $(M|N)$ can be computed in $\mathcal{O}(l^3)$ time.

Lemma 4

If $(M|N) = M^{\frac{N-1}{2}} \mathrm{mod} N \ \forall M \in \Phi(N)$*, then* N *is a prime.*

Theorem 3

If N *is an odd composite, then for at least half of* $M \in \Phi(N)$*,* $(M|N) \neq M^{\frac{N-1}{2}} \mathrm{mod} N$*.*

These two results suggest a Monte Carlo algorithm for compositeness. Given an odd integer N, this is the algorithm:

> Generate a random integer M between 2 and $N-1$ and calculate (M, N).
>
> If $(M, N) > 1$, reply "N is a composite".
>
> Otherwise calculate $(M|N)$, $M^{\frac{N-1}{2}} \mathrm{mod} N$ and compare;
>
> > if they are not equal, reply "N is a composite", otherwise reply "N is probably a prime".

Randomized Complexity Classes

Definition 3

Let N be a polynomial-time bounded nondeterministic Turing machine. For all of its computations on input x it halts after the same number of steps, a polynomial in $|x|$. Assume that at each step there are exactly two nondeterministic choices.
Now, let L be a language. A **polynomial Monte Carlo Turing machine** *for L is a Turing machine, as above, with*

- *$p(n)$ steps in each computation on an input of length n and*
- *for each string x,*
  - *if $x \in L$, then at least half of the $2^{p(|x|)}$ computations halt with "yes" and*
  - *if $x \notin L$, then all computations halt with "no".*

The class RP

(Randomized Polynomial)

- $L \in \mathbf{RP}$ if:
  - $x \in L \Rightarrow \mathrm{P}(N(x) = \text{"yes"}) \geq \frac{1}{2} + \epsilon$
  - $x \notin L \Rightarrow \mathrm{P}(N(x) = \text{"no"}) = 1$
- $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$
- There can be false negatives but no false positives.
- Similarly we define **coRP**:
  $L \in \mathbf{coRP}$ if:
  - $x \notin L \Rightarrow \mathrm{P}(N(x) = \text{"no"}) \geq \frac{1}{2} + \epsilon$
  - $x \in L \Rightarrow \mathrm{P}(N(x) = \text{"yes"}) = 1$
- In **coRP** there can be false positives but no false negatives.
- PRIMES $\in$ **coRP**.

The class ZPP

(Zero-error Probabilistic Polynomial)

- **ZPP**=**RP** $\cap$ **coRP**
- A problem in **ZPP** has two Monte Carlo algorithms: one with no false positives and one with no false negatives.
- If we execute both algorithms k times independently, the probability of no definite answer is $\frac{1}{2^k}$.
- $L \in$ **ZPP** if:
  - $x \in L \Rightarrow \mathrm{P}(N(x) = \text{"no"}) = 0$
  - $x \notin L \Rightarrow \mathrm{P}(N(x) = \text{"yes"}) = 0$
  - $\exists\ \epsilon > 0$: $\mathrm{P}(N(x) = \mathrm{UNK}) < \epsilon$
- **ZPP** algorithms are called **Las Vegas** algorithms.

The class PP

- $L \in \textbf{PP}$ if, $\forall x$, $\exists\ \epsilon > 0$:
  - $x \in L \Rightarrow \mathrm{P}(N(x) = \text{"yes"}) \geq \frac{1}{2} + \epsilon$
  - $x \notin L \Rightarrow \mathrm{P}(N(x) = \text{"no"}) \geq \frac{1}{2} + \epsilon$
- We say that N decides L "by majority".
- $\epsilon > 0$ depends on the input x.
- Even if we run it polynomially many times, we can't decrease the error probability.
- MAJSAT is **PP**-complete.
- **PP** is closed under complement.

Theorem 4

$\mathbf{NP} \subseteq \mathbf{PP}$.

Proof:

Let $L \in \mathbf{NP}$, decided by a N.D.T.M. N. Construct a Turing machine N': N' is identical to N except that it has a new initial state and a nondeterministic choice out of it. One choice gets us to the ordinary computation of N. The other choice gets us to a computation that always accepts (with the same number of steps). On input x, N produces $2^{p(|x|)}$ computations ($p(|x|)$ steps). N' produces $2^{p(|x|)+1}$ computations. At least half of them halt with "yes". Thus, $x \in L \iff$ there is at least one computation of N that accepts $\iff N'$ accepts L by majority $\iff L \in \mathbf{PP}$.

$\square$

The class BPP

(Bounded Probabilistic Polynomial)

- Suppose that you have a biased coin (one side has probability $\frac{1}{2} + \epsilon$ to appear).
- How would you detect which of the two sides is more likely to appear?
- Flip the coin many times.
- How many times do you have to flip it in order to guess correctly with high probability?

Lemma 5 (The Chernoff Bound)

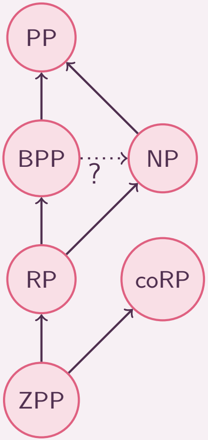
$x_1, \ldots, x_n$ independent random variables taking the values 0 and 1 with probabilities $1-p$ and p respectively. $X = \sum_{i=1}^{n} x_i$. Then $\forall\ 0 \leq \theta \leq 1$, $\mathrm{P}(X \geq (1+\theta)pn) \leq e^{-\frac{\theta^2}{3}pn}$.

Now by the Chernoff Bound, with $p = \frac{1}{2} + \epsilon$ and taking $\theta = \frac{\epsilon}{\epsilon + \frac{1}{2}}$ we have that $\mathrm{P}(X \leq \frac{n}{2}) \leq e^{-\frac{\epsilon^2 n}{6}}$.

Thus, we can detect which side is more likely to appear by flipping the coin about $\frac{1}{\epsilon^2}$ times and taking the majority.

- $L \in$ **BPP** if, $\forall x$:
  - $x \in L \Rightarrow \mathrm{P}(N(x) = \text{"yes"}) \geq \frac{3}{4}$
  - $x \notin L \Rightarrow \mathrm{P}(N(x) = \text{"no"}) \geq \frac{3}{4}$
- Equivalently, $L \in$ **PP** if, $\exists\ \epsilon > 0$, $\forall x$:
  - $x \in L \Rightarrow \mathrm{P}(N(x) = \text{"yes"}) \geq \frac{1}{2} + \epsilon$
  - $x \notin L \Rightarrow \mathrm{P}(N(x) = \text{"no"}) \geq \frac{1}{2} + \epsilon$
- In fact, we can take any number strictly between 0 and $\frac{1}{2}$ (e.g. $\frac{2}{3}$, $\frac{3}{5}$, $\ldots$).
- In this class, ϵ is not allowed to depend on the input x.
- We say that N decides L by "clear majority".
- After repeating polynomially many times we can decrease the error probability.
- **BPP** = **coBPP**

Class Overview



- $\mathbf{P} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$
- $\mathbf{NP} \subseteq \mathbf{PP}$
- $\mathbf{RP} \subseteq \mathbf{BPP} \subseteq \mathbf{PP}$
- $\mathbf{BPP} \overset{?}{\subseteq} \mathbf{NP}$

Random Sources

Definition 4

A **perfect random source** *is a random variable with values that are infinite sequences* $(x_1, x_2, \ldots)$ *of bits such that* $\forall\ n > 0$ *and* $\forall\ (y_1, \ldots, y_n) \in \{0, 1\}^n$ *we have* $P[x_i = y_i, i = 1, \ldots, n] = 2^{-n}$*.*

- A perfect random source must have *indpendence* and *fairness*.
- There are no perfect random sources in nature.
- The outcome of any physical random source tends to be affected by its previous outcomes.

Slightly Random Sources

Definition 5

*Let $0 < \delta < \frac{1}{2}$ and a function $p : \{0,1\}^* \to [\delta, 1-\delta]$. A δ-**random source** is a random variable with values that are infinite sequences and the probability that $(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ is given by*

$$\prod_{i=1}^{n} (y_i p(y_1 \ldots y_{i-1})) + (1 - y_i)(1 - p(y_1 \ldots y_{i-1})).$$

- p is a function completely unknown to us.
- If $\delta = \frac{1}{2}$ then we have a perfect random source.
- If $\delta < \frac{1}{2}$ then we say we have a **slightly random source**.

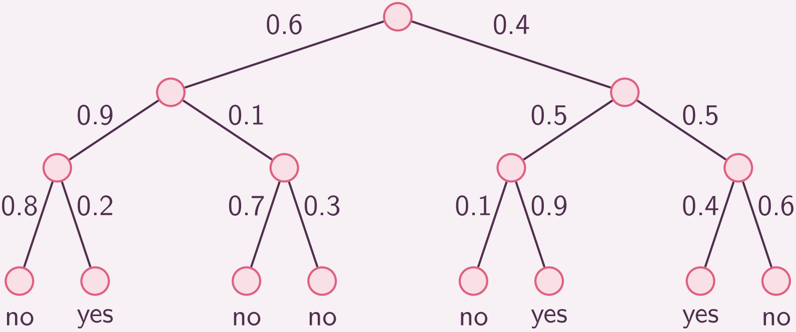
Since we know nothing about p we can't let a slightly random source run a randomized algorithm. But they can simulate any randomized algorithm with polynomial loss of efficiency.

Definition 6

*Let N be a Turing machine, standardized as previously, and $0 < \delta < \frac{1}{2}$. A δ-**assignment** F to $N(x)$ is a mapping from the edges of $N(x)$ to $[\delta, 1-\delta]$, such that the two edges leaving each internal node are assigned numbers adding up to 1.*

For each leaf l, the probability of l is $\prod_{a \in P[l] F(a)}$, where $P[l]$ is the path from root to leaf l.

Example: Computation tree and 0.1 assignment



$$\mathrm{P}(N(x) = \text{"yes"}) = 0.6 \cdot 0.9 \cdot 0.2 + 0.4 \cdot 0.5 \cdot 0.9 + 0.4 \cdot 0.5 \cdot 0.4$$

Definition 7

*A language L is in δ-**RP** if there is a N.D.T.M. N, as above, such that:*

- $x \in L \Rightarrow P(N(x) = \text{"yes"} | F) \geq \frac{1}{2}$ *and*
- $x \notin L \Rightarrow P(N(x) = \text{"yes"} | F) = 0$, *for all δ-assignments F.*

*A language L is in δ-**BPP** if there is a N.D.T.M. N, as above, such that:*

- $x \in L \Rightarrow P(N(x) = \text{"yes"} | F) \geq \frac{3}{4}$ *and*
- $x \notin L \Rightarrow P(N(x) = \text{"no"} | F) \geq \frac{3}{4}$, *for all δ-assignments F.*

- 0-**RP**=0-**BPP**=**P**
- $\frac{1}{2}$-**RP**=**RP**
- $\frac{1}{2}$-**BPP**=**BPP**

Theorem 5

*For any $\delta > 0$, δ-***BPP**=**BPP**.

Corollary 1

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

Circuit Complexity

Definition 8

1. **Size** *of a circuit: the numbers of gates in it.*
2. **Family of circuits***: an infinite sequence* $\mathcal{C} = (C_0, C_1, \ldots)$ *of Boolean circuits:* C_n *has* n *input variables.*
3. $L \subseteq \{0,1\}^*$ *has* **polynomial circuit***:* $\exists\, \mathcal{C} = (C_0, C_1, \ldots)$*:*
   a) *the size of* C_n *is at most* $p(n)$*,* p *fixed polynomial,*
   b) $\forall\, x \in \{0,1\}^*$, $x \in L$ *iff* $C_{|x|}$*'s output is true.*

What kinds of languages have polynomial circuits?

REACHABILITY has a polynomial circuit.

Proposition 1

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

Proof:

For each $L \in \mathbf{P}$ decided in time $p(n)$ and for each input x there is a variable-free circuit with $\mathcal{O}(p(|x|)^2)$ gates: output is true iff $x \in L$ (Theorem 8.1). When $L \subseteq \{0,1\}^*$, we can modify the input gates so that they are variables reflecting the symbols of x. □

The converse fails:

There are undecidable languages that have polynomial circuits.

(e.g. $L \subseteq \{0,1\}^*$ undecidable, $U \subseteq \{1\}^*$,
$U = \{1^n : \text{binary expansion of } n \text{ in } L\}$. U is undecidable but has a polynomial circuit.)

Definition 9

- $\mathcal{C} = (C_0, C_1, \ldots)$ *is* **uniform***: there is a* $\log n$*-space bounded Turing machine which on input* 1^n *outputs* C_n*.*
- L *has* **uniformly polynomial circuits***: there is a uniform family* $\mathcal{C}$ *that decides* L*.*

Theorem 6

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

Proof:

($\Leftarrow$) If $L \in \mathbf{P}$, the construction of C_n can be done in $\mathcal{O}(\log n)$ space (Theorem 8.1).
($\Rightarrow$) If L has a uniformly polynomial family of circuits, then we can build $C_{|x|}$ in $\log|x|$ space, hence in polynomial time. Then we can evaluate it in polynomial time. $\square$

The last Theorem relates to the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem. Indeed, the $\mathbf{P} \neq \mathbf{NP}$ conjecture is equivalent to:

A **NP**-complete problems have no uniformly polynomial circuits.

B **NP**-complete problems have no polynomial circuits.

Thus, showing that a specific **NP**-complete problem has no polynomial circuits implies $\mathbf{P} \neq \mathbf{NP}$.

The last result suggests that circuits are useless in proving that $\mathbf{P} \neq \mathbf{BPP}$:

Theorem 7

All languages in **BPP** *have polynomial circuits.*

Proof idea:

Let $L \in \mathbf{BPP}$. $\forall$ n we can construct C_n based on a sequence $A_n = (a_1, \ldots, a_m)$, $a_i \in \{0,1\}^{p(n)}$, $p(n)$ the lenght of the computations of N.D.T.M. N that decides L by clear majority, $m = 12(n+1)$. Each a_i represents a possible sequence of choices for N. C_n simulates N with each a_i and takes the majority of the outcomes.

It can be proven that: $\forall$ $n > 0$, $\exists$ a set A_n of $m = 12(n+1)$ bitstrings: $\forall$ input x, $|x| = n$, fewer than half of the choices in A_n are bad.

Now, given such an A_n we can build a circuit C_n with $\mathcal{O}(n^2 p^2(n))$ gates that simulate N and then takes the majority of the outcomes.

$\square$