TECHNISCHE UNIVERSITÄT DRESDEN

FOUNDATIONS OF COMPLEXITY THEORY

Lecture 17: Probabilistic Turing Machines

David Carral
Knowledge-Based Systems

TU Dresden, February 1, 2021

Randomness in Computation

Random number generators are an important tool in programming

- Many known algorithms use randomness
- DTMs are fully deterministic without random choices
- NTMs have choices, but are not governed by probabilities

Could a Turing machine benefit from having access to (truly) random numbers?

Example: Finding the Median

It is of interest to select the k-th smallest element of a set of numbers.

For example, the median of a set of numbers $\{a_1, \ldots, a_n\}$ is the $\lceil \frac{n}{2} \rceil$-th smallest number.

(Note: we restrict to odd n and disallow repeated numbers for simplicity)

The following simple algorithm selects the k-th smallest element:

```
01 SELECTKTHELEMENT($k$, $a_1$,...,$a_n$) :
02   pick some $p \in \{1, \ldots, n\}$  // select pivot element
03   $c$ := number of elements $a_i$ such that $a_i \leq a_p$
04   if $c == k$ :
05     return $a_p$
06   else if $c > k$ :
07     $L$ := list of all $a_i$ with $a_i < a_p$
08     return SELECTKTHELEMENT(k,L)
09   else if $c < k$ :
10     $L$ := list of all $a_i$ with $a_i > a_p$
11     return SELECTKTHELEMENT(k-c,L)
```

Example: Finding the Median – Analysis (1)

```
01 SELECTKTHELEMENT($k$, $a_1$,...,$a_n$) :
02   pick some $p \in \{1, \ldots, n\}$  // select pivot element
03   $c$ := number of elements $a_i$ such that $a_i \leq a_p$
04   if $c == k$ :
05     return $a_p$
06   else if $c > k$ :
07     $L$ := list of all $a_i$ with $a_i < a_p$
08     return SELECTKTHELEMENT(k,L)
09   else if $c < k$ :
10     $L$ := list of all $a_i$ with $a_i > a_p$
11     return SELECTKTHELEMENT(k-c,L)
```

What is the runtime of this algorithm?

- Lines 03, 07, and 10 run in $O(n)$
- The considered set shrinks by at least one element per iteration: $O(n)$ iterations

$\leadsto$ In the worst case, the algorithm requires quadratic time
So it would be faster to sort the list in $O(n \log n)$ and look up the k-th smallest element directly!

Example: Finding the Median – Analysis (2)

```
SELECTKTHELEMENT($k$, $a_1$, ..., $a_n$) :
  pick some $p \in \{1, \ldots, n\}$  // select pivot element
  $c$ := number of elements $a_i$ such that $a_i \leq a_p$
  if $c == k$ :
    return $a_p$
  else if $c > k$ :
    $L$ := list of all $a_i$ with $a_i < a_p$
    return SELECTKTHELEMENT(k,L)
  else if $c < k$ :
    $L$ := list of all $a_i$ with $a_i > a_p$
    return SELECTKTHELEMENT(k-c,L)
```

However, what if we pick pivot elements at random with uniform probability?

- then it is extremely unlikely that the worst case occurs
- one can show that the expected runtime is linear [Arora & Barak, Section 7.2.1]
- worse than linear runtimes can occur, but the total probability of such runs is 0

The algorithm runs in almost certain linear time.

A refined implementation that works with repeated numbers is Quickselect.

Probabilistic Turing Machines

How can we incorporate the power of true randomness into Turing machine definition?

Definition 17.1: A probabilistic Turing machine (PTM) is a Turing machine with two deterministic transition functions, δ_0 and δ_1.
A run of a PTM is a TM run that uses either of the two transitions in each step.

- PTMs therefore are very similar to NTMs with (at most) two options per step
- We think of transitions as being selected randomly, with equal probability of 0.5: the PTM flips a fair coin in each step
- A DTM is a special PTM where both transition functions are the same

Example 17.2: The task of picking a random pivot element $p \in \{1, \ldots, n\}$ with uniform probability can be achieved by a PTM:

(1) Perform ℓ coin flips, where ℓ is the least number with $2^\ell \geq n$

(2) Each outcome $\{1, \ldots, n\}$ corresponds to one combination of the ℓ flips

(3) For any other combination (if $n \neq 2^\ell$): goto (1) Note that the probability of infinite repetition is 0.

The Language of a PTM

Under which condition should we say "w is accepted by the PTM $\mathcal{M}$"?

Some options: w is accepted by the PTM $\mathcal{M}$ if ...

(1) it is possible that it will halt and accept

(2) it is more likely than not that it will halt and accept

(3) it is more likely than, say, 0.75 that it will halt and accept

(4) it is certain that it will halt and accept (probability 1)

Main question: Which definition is needed to obtain practical algorithms?

- (1) corresponds to the usual acceptance condition for NTMs.
- (4) corresponds to the usual acceptance condition for "co-NTMs".
- (2) is similarly difficult to check (majority vote over all runs).
- (3) could be useful for determining $w \in \mathbf{L}(\mathcal{M})$ with high probability, but how would we know if $w \notin \mathbf{L}(\mathcal{M})$?

$\leadsto$ Definitions do not seem to capture practical & efficient probabilistic algorithms yet

Random numbers as witnesses

Towards efficient probabilistic algorithms, we can restrict to PTMs where any run is guaranteed to be of polynomial length.

A useful alternative view on such PTMs is as follows:

Definition 17.3 (Polytime PTM, alternative definition): A polynomially time-bounded PTM is a polynomially time-bounded deterministic TM that receives inputs of the form $w\#r$, where $w \in \Sigma^*$ is an input word, and $r \in \{0, 1\}^*$ is a sequence of random numbers of length polynomial in $|w|$. If $w\#r$ is accepted, we may call r a witness for w.

Note the similarity to the notion of polynomial verifiers used for NP.

The prior definition is closely related to the alternative version:

- Every run of a PTM corresponds to a sequence of results of coin flips
- Polytime PTMs only perform a polynomially bounded number of coin flips
- A DTM can simulate the same computation when given the outcome of the coin flips as part of the input

(Note: the polynomial bound comes from a fixed polynomial for the given TM, of course)

PP: Polynomial Probabilistic Time

Polynomial Probabilistic Time

The challenge of defining practical algorithms is illustrated by a basic class of PTM languages based on polynomial time bounds:

Definition 17.4: A language **L** is in Polynomial Probabilistic Time (PP) if there is a PTM $\mathcal{M}$ such that:

- there is a polynomial function f such that $\mathcal{M}$ will always halt after $f(|w|)$ steps on all input words w,
- if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] > \frac{1}{2}$,
- if $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{2}$.

Alternative view: We could also say that $\mathcal{M}$ is a polynomially time-bounded PTM that accepts any word that is accepted in the majority of runs (or: the majority of witnesses)
$\leadsto$ PP is sometimes called Majority-P (which would indeed be a better name)

PP is hard (1)

It turns out that PP is far from capturing the idea of "practically efficient":

Theorem 17.5: NP $\subseteq$ PP

Proof: Since DTMs are special cases of PTMs, $\mathbf{L_1} \in$ PP and $\mathbf{L_2} \leq_m \mathbf{L_1}$ imply $\mathbf{L_2} \in$ PP. It therefore suffices to show that some NP-complete problem is in PP.

The following PP algorithm $\mathcal{M}$ solves **Sat** on input formula φ:

(1) Randomly guess an assignment for φ.

(2) If the assignment satisfies φ, accept.

(3) If the assignment does not satisfy φ, randomly accept or reject with equal probability.

Therefore:

- if φ is unsatisfiable, $\Pr[\mathcal{M} \text{ accepts } \varphi] = \frac{1}{2}$: the input is rejected;
- if φ is satisfiable, $\Pr[\mathcal{M} \text{ accepts } \varphi] > \frac{1}{2}$: the input is accepted. $\square$

Complementing PP (1)

Theorem 17.6: PP is closed under complement.

Proof: Let $\mathbf{L} \in$ PP be accepted by PTM $\mathcal{M}$, time-bounded by the polynomial $p(n)$. We therefore know:

- If $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] > \frac{1}{2}$
- If $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{2}$

We first ensure that, in the second case, no word is accepted with probability $\frac{1}{2}$.

We construct an PTM $\mathcal{M}'$ that first executes $\mathcal{M}$, and then:

- if $\mathcal{M}$ rejects: $\mathcal{M}'$ rejects
- if $\mathcal{M}$ accepts: $\mathcal{M}'$ flips coins for $p(n) + 1$ steps, rejects if they all of these coins are heads, and accepts otherwise.

This gives us $\Pr[\mathcal{M}' \text{ accepts } w] = \Pr[\mathcal{M} \text{ accepts } w] - (\frac{1}{2})^{p(n)+1}$ for all $w \in \Sigma^*$.

We will show that $\mathcal{M}'$ still describes the language **L**.

Complementing PP (2)

Theorem 17.7: PP is closed under complement.

Proof (continued): $\Pr[\mathcal{M}' \text{ accepts } w] = \Pr[\mathcal{M} \text{ accepts } w] - (\frac{1}{2})^{p(n)+1}$. We claim:

- If $w \in \mathbf{L}$, then $\Pr[\mathcal{M}' \text{ accepts } w] > \frac{1}{2}$
- If $w \notin \mathbf{L}$, then $\Pr[\mathcal{M}' \text{ accepts } w] < \frac{1}{2}$

The second inequality is clear (we subtract a non-zero number from $\leq \frac{1}{2}$).

The first inequality follows since the probability of any run of $\mathcal{M}$ on inputs of length n is an integer multiple of $(\frac{1}{2})^{p(n)}$. The same holds for sums of probabilities of runs, hence, if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \geq \frac{1}{2} + (\frac{1}{2})^{p(n)}$. The claim follows since $(\frac{1}{2})^{p(n)} > (\frac{1}{2})^{p(n)+1}$.

To finish the proof, we construct the complement $\overline{\mathcal{M}'}$ of $\mathcal{M}'$ by exchanging accepting and non-accepting states in $\mathcal{M}'$. Then:

- If $w \in \mathbf{L}$, then $\Pr\left[\overline{\mathcal{M}'} \text{ accepts } w\right] < \frac{1}{2}$
- If $w \notin \mathbf{L}$, then $\Pr\left[\overline{\mathcal{M}'} \text{ accepts } w\right] > \frac{1}{2}$

as required. □

PP is hard (2)

Since NP ⊆ PP (Theorem 17.5), we also get:

Corollary 17.8: coNP ⊆ PP

PP therefore appears to be strictly harder than NP or coNP.

The following strong result also hints in this direction:

Theorem 17.9: PH ⊆ P^{PP}

Note: The proof is based on a non-trivial result known as Toda's Theorem, which is about complexity classes where one can count satisfying assignments of propositional formulae ("**#Sat**"), together with the insight that this count can be computed in polynomial time using a PP oracle.

An upper bound for PP

We can also find a suitable upper bound for PP:

Theorem 17.10: PP ⊆ PSpace

Proof: Consider a PTM $\mathcal{M}$ that runs in time bounded by the polynomial $p(n)$.

We can decide if $\mathcal{M}$ accepts input w as follows:

(1) for each word $r \in \{0, 1\}^{p(|w|)}$:

(2) decide if $\mathcal{M}$ has an accepting run on w for the sequence r of random numbers;

(3) accept if the total number of accepting runs is greater than $2^{p(|w|)-1}$, else reject.

This algorithm runs in polynomial space, as each iteration only needs to store r and the tape of the simulated polynomial TM computation. □

Complete problems for PP

We can define PP-hardness and PP-completeness using polynomial many-one reductions as before.

Using the similarity with NP, it is not hard to find a PP-complete problem:

MajSat

Input:	A propositional logic formula φ.
Problem:	Is φ satisfied by more than half of its assignments?

It is not hard to reduce the question whether a PTMs accepts an input to **MajSat**:

- Describe the behaviour of the PTM in logic, as in the proof of the Cook-Levin Theorem
- Each satisfying assignment then corresponds to one run

BPP: A practical probabilistic class

How to use PTMs in practice

A practical idea for using PTMs:

- The output of a PTM on a single (random) run is governed by probabilities
- We can repeat the run many times to be more certain about the result

Problem: The acceptance probability for words in languages in PP can be arbitrarily close to $\frac{1}{2}$:

- It is enough if $2^{m-1} + 1$ runs accept out of 2^m runs overall
- So one would need an exponential number of repetitions to become reasonably certain

$\rightsquigarrow$ Not a meaningful way of doing probabilistic computing

We would rather like PTMs to accept with a fixed probability that does not converge to $\frac{1}{2}$.

A practical probabilistic class

The following way of deciding languages is based on a more easily detectable difference in acceptance probabilities:

Definition 17.11: A language **L** is in Bounded-Error Polynomial Probabilistic Time (BPP) if there is a PTM $\mathcal{M}$ such that:

- there is a polynomial function f such that $\mathcal{M}$ will always halt after $f(|w|)$ steps on all input words w,
- if $w \in \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \geq \frac{2}{3}$,
- if $w \notin \mathbf{L}$, then $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{3}$.

In other words: Languages in BPP are decided by polynomially time-bounded PTMs with error probability $\leq \frac{1}{3}$.

Note that the bound on the error probability is uniform across all inputs:

- For any given input, the probability for a correct answer is at least $\frac{2}{3}$
- It would be weaker to require that the probability of a correct answer is at least $\frac{2}{3}$ over the space of all possible inputs (this would allow worse probabilities on some inputs)

Better error bounds

Intuition suggests: If we run an PTM for a BPP language multiple times, then we can increase our certainty of a particular outcome.

Approach:

- Given input w, run $\mathcal{M}$ for k times
- Accept if the majority of these runs accepts, and reject otherwise.

Which outcome do we expect when repeating a random experiment k times?

- The probability of a single correct answer is $p \geq \frac{2}{3}$
- We therefore expect a percentage p of runs to return the correct result

What is the probability that we see some significant deviation from this expectation?

- It is still possible that only less than half of the runs return the correct result anyway
- How likely is this, depending on the number of repetitions k?

Chernoff bounds

Chernoff bounds are a general type of result for estimating the probability of a certain deviation from the expectation when repeating a random experiment.

There are many such bounds – some more accurate, some more usable. We merely give the following simplified special case:

Theorem 17.12: Let $X_1, \ldots, X_k$ be mutually independent random variables that can take values from $\{0, 1\}$, and let $\mu = \sum_{i=1}^{k} E[X_i]$ be the sum of their expected values. Then, for every constant $0 < \delta < 1$:

$$\Pr\left[\left|\sum_{i=1}^{k} X_i - \mu\right| \geq \delta\mu\right] \leq e^{-\delta^2\mu/4}$$

Example 17.13: Consider $k = 1000$ tosses of fair coins, $X_1, \ldots, X_{1000}$, with heads corresponding to result 1 and tails corresponding to 0. We expect $\mu = \sum_{i=1}^{n} E[X_i] = 500$ to be the sum of these experiments. By the above bound, the probability of seeing $600 = 500 + 0.2 \cdot 500$ or more heads is

$$\Pr\left[\left|\sum_{i=1}^{k} X_i - 500\right| \geq 100\right] \leq e^{-0.2^2 \cdot 500/4} \leq 0.0068.$$

Much better error bounds

We can now show that even a small, input-dependent probability of finding correct answers is enough to construct an algorithm whose certainty is exponentially close to 1:

Theorem 17.14: Consider a language **L** and a polynomially time-bounded PTM $\mathcal{M}$ for which there is a constant $c > 0$ such that, for every word $w \in \Sigma^*$, $\Pr[\mathcal{M}$ classifies w correctly$] \geq \frac{1}{2} + |w|^{-c}$.
Then, for every constant $d > 0$, there is a polynomially time-bounded PTM $\mathcal{M}'$ such that $\Pr[\mathcal{M}'$ classifies w correctly$] \geq 1 - 2^{-|w|^d}$.

Proof: We construct $\mathcal{M}'$ as before by running $\mathcal{M}$ for k times, where we set $k = 8|w|^{2c+d}$. Note that this is number of repetitions is polynomial in $|w|$.

To use our Chernoff bound, define k random variables X_i with $X_i = 1$ if the ith run of $\mathcal{M}$ returns the correct result:

- Set p to be $\Pr[X_i = 1] \geq \frac{1}{2} + |w|^{-c}$
- Then $E[\sum_{i=1}^{k} X_i] = pk$

Much better error bounds (continued)

We can now show that even a small, input-dependent probability of finding correct answers is enough to construct an algorithm whose certainty is exponentially close to 1:

Theorem 17.14: Consider a language **L** and a polynomially time-bounded PTM $\mathcal{M}$ for which there is a constant $c > 0$ such that, for every word $w \in \Sigma^*$, $\Pr[\mathcal{M}$ classifies w correctly$] \geq \frac{1}{2} + |w|^{-c}$.
Then, for every constant $d > 0$, there is a polynomially time-bounded PTM $\mathcal{M}'$ such that $\Pr[\mathcal{M}'$ classifies w correctly$] \geq 1 - 2^{-|w|^d}$.

Proof (continued): We are interested in the probability that at least half of the runs are correct. This can be achieved by setting $\delta = \frac{1}{2} \cdot |w|^{-c}$.

Our Chernoff bound then yields:

$$\Pr\left[\left|\sum_{i=1}^{k} X_i - pk\right| \geq \delta pk\right] \leq e^{-\delta^2 pk/4} = e^{-(\frac{1}{2}\cdot|w|^{-c})^2 pk/4} \leq e^{-\frac{1}{4|w|^{2c}}\cdot\frac{1}{2}\cdot 8|w|^{2c+d}} \leq e^{-|w|^d} \leq 2^{-|w|^d}$$

(where the estimations are dropping some higher-order terms for simplification).

BPP is robust

Theorem 17.14 gives a massive improvement in certainty at only polynomial cost. As a special case, we can apply this to BPP (where probabilities are fixed):

Corollary 17.15: Defining the class BPP with any bounded error probability $< \frac{1}{2}$ instead of $\frac{1}{3}$ leads to the same class of languages.

Corollary 17.16: For any language in BPP, there is a polynomial time algorithm with exponentially low probability of error.

BPP might be better than P for describing what is “tractable in practice.”

Understanding BPP

BPP is practical

We found (Theorem 21.12):

- If a polytime PTM produces the correct output with probability $\geq \frac{1}{2} + |w|^{-c}$,
- then some polytime PTM produces the correct output with probability $\geq 1 - 2^{-|w|^d}$.

In words: even a weak bound on the error is enough to obtain almost arbitrary certainty in polynomial time!

Corollary 21.15: Defining the class BPP with any bounded error probability $< \frac{1}{2}$ instead of $\frac{1}{3}$ leads to the same class of languages.

Corollary 21.16: For any language in BPP, there is a polynomial time algorithm with exponentially low probability of error.

BPP might be better than P for describing what is “tractable in practice”!

Summary and open questions

We have already seen that BPP is robust against the actual error bound

Moreover, it is not hard to show the following:

- BPP is closed under complement (exercise)
- $\mathrm{BPP}^{\mathrm{BPP}} = \mathrm{BPP}$ (exercise)

We have not discussed several important questions:

- What happens if we assume unfair coins? $(\Pr[\text{heads}] \neq \frac{1}{2})$
- How does BPP relate to other complexity classes?
- Which problems are in BPP and which are BPP-complete?

Robustness using unfair coins (1)

Would a PTM have greater power if its random number generator would output 1 with probability $\rho \neq \frac{1}{2}$?

Proposition 17.17: A coin with $\Pr[\text{heads}] = \rho$ can be simulated by a PTM in expected time $O(1)$ provided that the ith bit of ρ is computable in polynomial time w.r.t. i.

Proof: Let $0.\rho_1\rho_2\rho_3\cdots$ be the binary expansion of ρ. Starting with $i = 1$, do:

- Compute a random bit $b_i \in \{0, 1\}$
- If $\rho_i > b_i$, return “heads”
- If $\rho_i < b_i$, return “tails”
- If $\rho_i = b_i$, increment i and repeat procedure.

Analysis:

- The simulation reaches step $i + 1$ with probability $(\frac{1}{2})^i$
- Combined probability of “heads”: $\sum_i \rho_i \frac{1}{2^i} = \rho$
- The expected runtime is $O(\sum_i i^c \frac{1}{2^i})$, where c is a constant degree capturing the polynomial effort of computing ρ_i – this can be shown to be in $O(1)$. □

Robustness using unfair coins (2)

Note: Proposition 17.17 requires ρ to be efficiently computable. Unfair coins with hard to compute probabilities would indeed increase the computational power.

Conversely, we may ask if a PTM with unfair coin could simulate a fair coin:

Proposition 17.18: A coin with $\Pr[\text{heads}] = \frac{1}{2}$ can be simulated by a TM that may use a coin with heads-probability ρ in time $O(\frac{1}{\rho(1-\rho)})$.

Proof: See exercise (for the basic technique of simulating fair coins with arbitrary ones)

Note that the previous result does not require ρ to be computable.

Conclusion: BPP is rather robust against the use of different coins.

Polynomial Identity Testing

A problem in BPP

We give an example of a problem in BPP that is not known to be in P.

Polynomial Identity Testing (PIT):

- Task: Determine if two polynomial functions are equal, i.e., have the same results on all inputs
- The polynomials can be multivariate (i.e., contain more than two variables)
- Challenge: The polynomials are not given in their normal form (as a sum of monomials)

Approach: Reduce the question "$f = g$?" to the question "$f - g = 0$?," i.e., to the question if a given polynomial is equal to zero.

Example 17.19: We may ask if $(x+y)(x-y)$ equals $x^2 - y^2$. To answer this, we can test if the polynomial function $(x+y)(x-y) - (x^2 - y^2)$ equals zero.

Algebraic circuits and ZeroP

The representation we assume for polynomials in PIT are algebraic circuits:

- Algebraic circuits are like Boolean circuits but operate on integer numbers
- Gates perform arithmetic operations $+$, $-$, and $\times$, or have constant output 1
- There is one output

Note: it is easy to express the difference of the functions encoded in two algebraic circuits

ZeroP

Input:	Algebraic circuit C
Problem:	Does C return 0 on all inputs?

How difficult is **ZeroP**?

Observation:

- Algebraic circuits can encode polynomials very efficiently:
  a small circuit can express a polynomial that is large when written in the usual form

Example 17.20: It is easy to find a circuit of size $2k$ for $\prod_{i=1}^{k}(x_i + y_i)$ (assuming binary fan-in for multiplication gates), but writing this function as a sum of monomials requires 2^k monomials of the form $z_1 \cdot z_2 \cdots z_k$ where $z_i \in \{x_i, y_i\}$.

- Nevertheless, the output of a circuit is easy to compute

Surprisingly (?): There is an efficient probabilistic algorithm for **ZeroP**

How frequently do non-zero polynomials compute zero?

The total degree of a (multivariate) monomial is the sum of the degrees of all of its variables, and the total degree of a polynomial is the maximal degree of its monomials.

The following property is the key to showing **ZeroP** $\in$ BPP:

Lemma 17.21 (Schwartz-Zippel Lemma): Consider a non-zero multivariate polynomial $p(x_1, \ldots, x_m)$ of total degree $\leq d$, and a finite set S of integers. If $a_1, \ldots, a_m$ are chosen randomly (with replacement) from S, then

$$\Pr\left[p(a_1, \ldots, a_m) = 0\right] \leq \frac{d}{|S|}$$

Proof: See Exercise.

A probabilistic algorithm for **ZeroP** (1)

By Schwartz-Zippel, we just need to randomly sample numbers from a large enough set S to find a non-zero value with high probability, namely $1 - \frac{d}{|S|}$.

What is the degree d of a polynomial encoded in an algebraic circuit?
A circuit of size n can compute degrees of at most 2^n.
$\rightsquigarrow$ for a set S of size $3 \cdot 2^n$, we expect a non-zero value with probability $\geq 1 - \frac{2^n}{3 \cdot 2^n} = \frac{2}{3}$

Algorithm: For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 3 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit to compute $p(a_1, \ldots, a_m)$
- Accept if $p(a_1, \ldots, a_m) = 0$ and reject otherwise.

Analysis: If $p \in$ **ZeroP**, the algorithm will always accept. Otherwise, if $p \notin$ **ZeroP**, it will reject with probability $\geq \frac{2}{3}$.

A probabilistic algorithm for **ZeroP** (2)

Did we show **ZeroP** $\in$ BPP?

There is a problem with our algorithm:

- We can sample the numbers a_i in polynomial time (polynomial number of bits)
- But if the degree of the polynomial is as high as 2^n, then the output can be as high as $(3 \cdot 2^n)^{2^n}$, requiring $O(2^n)$ bits to store!

One can solve this problem as follows:

Algorithm: For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select a number $k \in \{1, \ldots, 2^{2n}\}$
- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 10 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit modulo k to compute $p(a_1, \ldots, a_m) \mod k$
- Repeat this experiment for $4n$ times and accept if and only if the outcome is 0 in all cases

ZeroP $\in$ BPP

Algorithm (with fingerprinting): For a polynomial $p(x_1, \ldots, x_m)$

- Randomly select a number $k \in \{1, \ldots, 2^{2n}\}$
- Randomly select $a_1, \ldots, a_m \in \{1, \ldots, 10 \cdot 2^n\}$ (a total of $O(n \cdot m)$ random bits)
- Evaluate the circuit modulo k to compute $p(a_1, \ldots, a_m) \mod k$
- Repeat this experiment for $4n$ times and accept if and only if the outcome is 0 in all cases

Analysis: (additional details in Arora & Barak, Section 7.2.3)

- If $p(a_1, \ldots, a_m) = 0$ then $p(a_1, \ldots, a_m) = 0 \mod k$, so the algorithm surely accepts
- If $p(a_1, \ldots, a_m) \neq 0$ then $p(a_1, \ldots, a_m) \neq 0 \mod k$ if k does not divide $p(a_1, \ldots, a_m)$
- Claim: the probability of k dividing $p(a_1, \ldots, a_m)$ is $\leq \frac{1}{4n}$. Proof sketch:
  - We can restrict to cases where k (by random chance) is prime: for large n, there are at least $\frac{2^{2n}}{2n}$ prime numbers $\leq 2^{2n}$ (Prime Number Theorem)
  - A number has only logarithmically many prime factors ($O(n \cdot 2^n)$ in our case)
  - One can estimate that k with probability $\geq \frac{1}{4n}$ is both (i) a prime number and (ii) not among the prime factors of $p(a_1, \ldots, a_m)$ $\square$

Note: This does not yield a probability of error $\leq \frac{1}{3}$, but error probability $\leq \frac{1}{10} + \frac{9}{10}(1 - \frac{1}{4n})^{4n} \leq \frac{1}{10} + \frac{9}{10}\frac{1}{e} \leq 0.44$, which suffices.

Further problems in BPP

Other algorithms in BPP include:

- Testing for perfect matching in a bipartite graph
  Informally: checking whether every member of two equal-sized populations of heterosexual men and women can engage in monogamous partnerships according to their expressed preferences.
  - Can be reduced to checking if a variable matrix has non-zero determinant
  - Similar to **ZeroP**, one can use Schwartz-Zippel here
- Primality testing (**Primes**)
  - A classical probabilistic algorithm discovered in the 1970s
  - In 2002, Agrawal, Kayal, and Saxena found a deterministic polynomial algorithm
- "Monte-Carlo algorithms"
  - These are a general class of algorithms with "probably correct" output
  - BPP contains polynomial-time Monte-Carlo algorithms

Further probabilistic classes

Types of errors

We have defined BPP by restricting the probability of error to $\leq \frac{1}{3}$.

However, there are two types of errors:

- False positives: the PTM accepts a word that is not in the language
- False negatives: the PTM rejects a word that is in the language

Common BPP algorithms can often avoid one of these errors:

Example 17.22: Our previous algorithm for polynomial identity testing aimed to decide **ZeroP**. For inputs $w \in$ **ZeroP**, the algorithm accepted with probability 1 (no false negatives). Uncertainty only occurred for inputs $w \notin$ **ZeroP** (false positives were possible, though unlikely).

Randomised Polynomial Time

Excluding false positives/negatives from BPP leads to classes with one-sided error:

Definition 17.23: A language **L** is in Randomised Polynomial Time (RP) if there is a PTM $\mathcal{M}$ such that:

- there is a polynomial function f such that $\mathcal{M}$ will always halt after $f(|w|)$ steps on all input words w,
- if $w \in$ **L**, then $\Pr[\mathcal{M}$ accepts $w] \geq \frac{2}{3}$,
- if $w \notin$ **L**, then $\Pr[\mathcal{M}$ accepts $w] = 0$.

Definition 17.24: A language **L** is in coRP if its complement is in RP, i.e., if there is a polynomially time-bounded PTM $\mathcal{M}$ such that:

- if $w \in$ **L**, then $\Pr[\mathcal{M}$ accepts $w] = 1$,
- if $w \notin$ **L**, then $\Pr[\mathcal{M}$ accepts $w] \leq \frac{1}{3}$.

Example 17.25: **ZeroP** $\in$ coRP.

Probability amplification for RP and coRP

It is clear from the definitions that RP $\subseteq$ BPP and coRP $\subseteq$ BPP.

Hence, we can apply Theorem 21.14 to amplify the output probability.

However, the situation for one-sided error classes is actually much simpler:

Theorem 17.26: Consider a language **L** and a polynomially time-bounded PTM $\mathcal{M}$ for which there is a constant $c > 0$ such that, for every word $w \in \Sigma^*$,

- if $w \in$ **L** then $\Pr[\mathcal{M}$ accepts $w] \geq |w|^{-c}$
- if $w \notin$ **L** then $\Pr[\mathcal{M}$ accepts $w] = 0$

Then, for every constant $d > 0$, there is a polynomially time-bounded PTM $\mathcal{M}'$ such that

- if $w \in$ **L** then $\Pr[\mathcal{M}'$ accepts $w] \geq 1 - 2^{-|w|^d}$
- if $w \notin$ **L** then $\Pr[\mathcal{M}'$ accepts $w] = 0$.

Proof: Much simpler than for BPP (exercise). □

RP and NP

The asymmetric acceptance conditions of RP reminds us of NP, since already "some" accepting runs are enough to prove acceptance.

Indeed, we get:

Theorem 17.27: RP $\subseteq$ NP

Proof: If $\mathcal{M}$ satisfies the RP acceptance conditions for **L**, then $\mathcal{M}$ can be considered as an NTM that accepts **L** with respect to the usual non-deterministic acceptance conditions. Indeed, $\mathcal{M}$ has an accepting run on input $|w|$ if and only if $w \in$ **L**. □

Similarly, we find coRP $\subseteq$ coNP.

Recall: While RP $\subseteq$ BPP, we do not know whether BPP $\subseteq$ NP.

Zero-sided error

Instead of admitting a possibly false answer (positive or negative), one can also require the correct answer while making some concessions on runtime:

Definition 17.28: A PTM $\mathcal{M}$ has expected runtime $f : \mathbb{N} \to \mathbb{R}$ if, for any input w, the expectation $E[T_w]$ of the number T_w of steps taken by $\mathcal{M}$ on input w is $T_w \leq f(|w|)$.

ZPP is the class of all languages for which there is a PTM $\mathcal{M}$ that

- returns the correct answer whenever it halts,
- has expected runtime f for some polynomial function f.

ZPP is for zero-error probabilistic polynomial time.

Note: In general, algorithms that produce correct results while giving only probabilistic guarantees on resource usage are called Las Vegas algorithms, as opposed to Monte Carlo algorithms, which have guaranteed resource bounds but probabilistic correctness (as in the case of BPP).

Zero-sided vs. one-sided error

In spite of the different approaches of expected error vs. expected runtime, we find a close relation between ZPP, RP, and coRP:

Theorem 17.29: ZPP = RP ∩ coRP

Proof: ZPP ⊆ RP: Given a ZPP algorithm $\mathcal{M}$, construct an RP algorithm by running $\mathcal{M}$ for three times the expected (polynomial) runtime t. If it stops, return the same answer; if it times out, reject.

- For any random variable X and $c > 0$, Markov's inequality implies: $\Pr[X \geq cE[X]] \leq \frac{E[X]}{cE[X]} = \frac{1}{c}$
- Hence the probability of $\mathcal{M}$ running for $\geq 3t$ is $\leq \frac{1}{3}$
- Therefore, the probability of a false negative (due to a timeout) is $\leq \frac{1}{3}$

ZPP ⊆ coRP is dual; we just have to accept after timeout.

Zero-sided vs. one-sided error

In spite of the different approaches of expected error vs. expected runtime, we find a close relation between ZPP, RP, and coRP:

Theorem 17.29: ZPP = RP ∩ coRP

Proof: ZPP ⊇ RP ∩ coRP: Assume we have an RP algorithm $\mathcal{A}$ and a coRP algorithm $\mathcal{B}$ for the same language **L**. To obtain a ZPP algorithm, we run $\mathcal{A}$ and $\mathcal{B}$ on input w:

- If $\mathcal{A}$ accepts, accept
- If $\mathcal{B}$ rejects, reject
- If $\mathcal{A}$ rejects and $\mathcal{B}$ accepts, repeat the experiment.

Since RP has no false positives and coRP has no false negatives, this can only return the correct answer.

The probability of repetition is $\leq \frac{1}{3}$, since it requires one of the algorithms to be in error.

Hence the probability of k repetitions is $\leq 3^{-k}$, for an expected runtime of $\leq \sum_{k\geq 0} \frac{(k+1)p}{3^k}$, where p is the combined (polynomial) runtime of $\mathcal{A}$ and $\mathcal{B}$. This is polynomial. □

Summary and Outlook

Probabilistic TMs can be used to randomness in computation

PP defines a simple "probabilistic" class, but is too powerful in practice.

BPP provides a robust notion of practical probabilistic algorithm

Probabilistic classes with ones-sided error – RP and coRP – are common.

ZPP defines random computations with zero-sided error, but probabilistic runtime.

What's next?

- Oral Exams: 24th of February
- Deadline for Exam Registration: 15th of February
- Examinations