TECHNISCHE UNIVERSITÄT DRESDEN

# COMPLEXITY THEORY

## Lecture 21: Probabilistic Turing Machines

**Markus Krötzsch**

**Knowledge-Based Systems**

TU Dresden, 13th Jan 2020

# 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

# 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)

# 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:

```
SELECTKTHELEMENT(k, a_1, ..., a_n) :
   pick some p ∈ {1, ..., n}   // select pivot element
   c := number of elements a_i such that a_i ≤ 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)
```

# Example: Finding the Median – Analysis (1)

```
SELECTKTHELEMENT(k, a_1, ..., a_n) :
   pick some p ∈ {1, ..., n}   // select pivot element
   c := number of elements a_i such that a_i ≤ 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)
```

What is the runtime of this algorithm?

# Example: Finding the Median – Analysis (1)

```
01 selectKthElement(k, a_1,...,a_n) :
02    pick some p ∈ {1,...,n}   // select pivot element
03    c := number of elements a_i such that a_i ≤ 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

# Example: Finding the Median – Analysis (1)

```
01 selectKthElement(k, a_1,...,a_n) :
02    pick some p ∈ {1,...,n}  // select pivot element
03    c := number of elements a_i such that a_i ≤ 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 ∈ {1,...,n}   // select pivot element
   c := number of elements a_i such that a_i ≤ 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?

# Example: Finding the Median – Analysis (2)

```
SELECTKTHELEMENT(k, a_1,...,a_n) :
   pick some p ∈ {1,...,n}   // select pivot element
   c := number of elements a_i such that a_i ≤ 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?

# Probabilistic Turing Machines

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

**Definition 21.1:** A probabilistic Turing machine (PTM) is a Turing machine with two deterministic transition functions, $\delta_0$ and $\delta_1$.
A run of a PTM is a TM run that uses either of the two transitions in each step.

# Probabilistic Turing Machines

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

**Definition 21.1:** A probabilistic Turing machine (PTM) is a Turing machine with two deterministic transition functions, $\delta_0$ and $\delta_1$.
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

# Probabilistic Turing Machines

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

**Definition 21.1:** A probabilistic Turing machine (PTM) is a Turing machine with two deterministic transition functions, $\delta_0$ and $\delta_1$.
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 21.2:** The task of picking a random pivot element $p \in \{1, \ldots, n\}$ with uniform probability can be achieved by a PTM:

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

(2) Each outcome $\{1, \ldots, n\}$ corresponds to one combination of the $\ell$ 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 $\mathcal{M}$"?

# 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?

# 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.

# 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".

# 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).

# 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 $w \notin \mathbf{L}(\mathcal{M})$?

# 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 21.3 (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.

# 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 21.3 (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 21.4:** 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 $\Pr[\mathcal{M} \text{ accepts } w] \leq \frac{1}{2}$.

# Polynomial Probabilistic Time

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

**Definition 21.4:** 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 21.5:** $\mathsf{NP} \subseteq \mathsf{PP}$

# PP is hard (1)

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

**Theorem 21.5:** 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.

# PP is hard (1)

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

**Theorem 21.5:** 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 $\varphi$:

(1) Randomly guess an assignment for $\varphi$.

(2) If the assignment satisfies $\varphi$, accept.

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

# PP is hard (1)

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

**Theorem 21.5:** 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 $\varphi$:

(1) Randomly guess an assignment for $\varphi$.

(2) If the assignment satisfies $\varphi$, accept.

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

Therefore:

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

# Complementing PP (1)

**Theorem 21.6:** PP is closed under complement.

# Complementing PP (1)

**Theorem 21.6:** PP is closed under complement.

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

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

# Complementing PP (1)

**Theorem 21.6:** PP is closed under complement.

**Proof:** Let **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 21.7:** 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 $\leq \frac{1}{2}$).

# Complementing PP (2)

**Theorem 21.7:** 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 $(\frac{1}{2})^{p(n)} > (\frac{1}{2})^{p(n)+1}$.

# Complementing PP (2)

**Theorem 21.7:** 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 $\subseteq$ PP (Theorem 21.5), we also get:

**Corollary 21.8:** coNP $\subseteq$ PP

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

# PP is hard (2)

Since NP $\subseteq$ PP (Theorem 21.5), we also get:

**Corollary 21.8:** coNP $\subseteq$ PP

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

The following strong result also hints in this direction:

**Theorem 21.9:** PH $\subseteq \mathrm{P}^{\mathrm{PP}}$

**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:

# An upper bound for PP

We can also find a suitable upper bound for PP:

**Theorem 21.10:** PP $\subseteq$ PSpace

# An upper bound for PP

We can also find a suitable upper bound for PP:

**Theorem 21.10:** PP $\subseteq$ 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 $r \in \{\mathtt{0}, \mathtt{1}\}^{p(|w|)}$:

(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 $\varphi$. |
| Problem: | Is $\varphi$ 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

# 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

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

# 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

$\leadsto$ 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 21.11:** 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.

# 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?

# 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

# 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 21.12:** 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 $0 < \delta < 1$:

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

# 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 21.12:** 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 21.13:** 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 21.14:** 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$] \geq 1 - 2^{-|w|^d}$.

# 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 21.14:** 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 $i$th 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 21.14:** 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 $w \in \Sigma^*$, $\Pr[\mathcal{M} \text{ classifies } w \text{ correctly}] \geq \frac{1}{2} + |w|^{-c}$.
Then, for every constant $d > 0$, there is a polynomially time-bounded PTM $\mathcal{M}'$ such that $\Pr[\mathcal{M}' \text{ classifies } w \text{ correctly}] \geq 1 - 2^{-|w|^d}$.

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

**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 21.14 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 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 is robust

Theorem 21.14 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 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 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 better definition of practical probabilistic algorithm

**What's next?**

- More probabilistic classes
- Quantum Computing
- Examinations