Computational Complexity Theory

Lecture 17: Probabilistic Turing Machines; Class BPP

Department of Computer Science,
Indian Institute of Science

Randomized computation

- So far, we have used deterministic TMs to model "real-world" computation. But, DTMs don't have the ability to make *random choices* during a computation.

- The usefulness of randomness in computation was realized as early as the 1940s when the first electronic computer, ENIAC, was developed.

Randomized computation

- So far, we have used deterministic TMs to model "real-world" computation. But, DTMs don't have the ability to make *random choices* during a computation.

- The usefulness of randomness in computation was realized as early as the 1940s when the first electronic computer, ENIAC, was developed.
  - The use of statistical methods in a computational model of a thermonuclear reaction for the ENIAC lead to the invention of the ***Monte Carlo methods***.

Randomized computation

- So far, we have used deterministic TMs to model "real-world" computation. But, DTMs don't have the ability to make *random choices* during a computation.

- The usefulness of randomness in computation was realized as early as the 1940s when the first electronic computer, ENIAC, was developed.

- To study randomized computation, we need to give TMs the power of generating random numbers.

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.

1 with probability ½
0 with probability ½

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.

$$X_{i+1} = aX_i + c \pmod{m}$$

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.

Square an n bit number to get a 2n bit number and take the middle n bits.

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are <u>*linear congruential generators*</u> and von Neumann's <u>*middle-square method*</u>.
- To what extent a PRG is adequate is studied under the topic `*Pseudorandomness*' in complexity theory.

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.
- We'll assume that a TM can generate, or has access to, truly random bits/coins. (We'll touch upon "truly vs biased random bits" at end of the lecture.)

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject).

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has <u>two</u> transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject). M runs in T(n) time if M always halts within T(|x|) steps <u>*regardless of its random choices*</u>.

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject). M runs in $T(n)$ time if M always halts within $T(|x|)$ steps *regardless of its random choices*.

- Note. PTMs and NTMs are syntatically similar – both have two transition functions.

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject). M runs in $T(n)$ time if M always halts within $T(|x|)$ steps *regardless of its random choices*.

- Note. But, semantically, they are quite different – unlike NTMs, PTMs are meant to model realistic computation devices.

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject). M runs in $T(n)$ time if M always halts within $T(|x|)$ steps *regardless of its random choices*.

- Note. The above definition allows a PTM M to not halt on some computation paths defined by its random choices (unless we explicitly say that M runs in $T(n)$ time). More on this later when we define ZPP.

Class BPP

- Definition. A PTM M *decides* a language L in time $T(n)$ if M runs in $T(n)$ time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in $\mathrm{BPTIME}(T(n))$ if there's PTM that decides L in $O(T(n))$ time.

Class BPP

- Definition. A PTM M *decides* a language L in time $T(n)$ if M runs in $T(n)$ time, and for every $x \in \{0,1\}^*$,
$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in $\mathrm{BPTIME}(T(n))$ if there's PTM that decides L in $O(T(n))$ time.

- Definition. $\mathrm{BPP} = \bigcup_{c>0} \mathrm{BPTIME}(n^c)$.

- Clearly, $\mathrm{P} \subseteq \mathrm{BPP}$.

Class BPP

- Definition. A PTM M *decides* a language L in time $T(n)$ if M runs in $T(n)$ time, and for every $x \in \{0,1\}^*$,

$$\underbrace{\Pr[M(x) = L(x)]}_{\text{Success probability}} \geq 2/3.$$

- Definition. A language L is in $\mathrm{BPTIME}(T(n))$ if there's PTM that decides L in $O(T(n))$ time.

- Definition. $\mathrm{BPP} = \bigcup_{c > 0} \mathrm{BPTIME}(n^c)$.

- Clearly, $\mathrm{P} \subseteq \mathrm{BPP}$.

Class BPP

- Definition. A PTM M *decides* a language L in time $T(n)$ if M runs in $T(n)$ time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in $\mathrm{BPTIME}(T(n))$ if there's PTM that decides L in $O(T(n))$ time.

- Definition. $\mathrm{BPP} = \bigcup_{c>0} \mathrm{BPTIME}(n^c)$.

- Clearly, $\mathrm{P} \subseteq \mathrm{BPP}$.

Remark. The defn of class BPP is robust. The class remains unaltered if we replace 2/3 by any constant ***strictly greater*** than (i.e., ***bounded away*** from) ½. We'll discuss this next.

Class BPP

- Definition. A PTM M *decides* a language L in time $T(n)$ if M runs in $T(n)$ time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in $\mathrm{BPTIME}(T(n))$ if there's PTM that decides L in $O(T(n))$ time.

- Definition. $\mathrm{BPP} = \bigcup_{c>0} \mathrm{BPTIME}(n^c)$.

  Bounded-error **P**robabilistic **P**olynomial-time

- Clearly, $P \subseteq \mathrm{BPP}$.

Remark. The defn of class BPP is robust. The class remains unaltered if we replace 2/3 by any constant ***strictly greater*** than (i.e., ***bounded away*** from) ½. We'll discuss this next.

Class BPP

- Definition. A PTM M *decides* a language L in time $T(n)$ if M runs in $T(n)$ time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in $\mathrm{BPTIME}(T(n))$ if there's PTM that decides L in $O(T(n))$ time.

- Definition. $\mathrm{BPP} = \bigcup_{c>0} \mathrm{BPTIME}(n^c)$.

- Clearly, $\mathrm{P} \subseteq \mathrm{BPP}$.

Remark. Achieving success probability ½ is trivial for any language. If we replace $\geq 2/3$ by $> ½$ then the corresponding class is called PP, which is (presumably) larger than BPP. More on PP later.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^{d})$.
- *Proof.* Let $|x| = n$. Think of M' that runs M on input x for $m = 4n^{2c+d}$ times independently. Let $b_1, \ldots, b_m$ be the outputs of these independent executions of M. M' outputs $\mathrm{Majority}(b_1, \ldots, b_m)$.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.
- *Proof.* Let $|x| = n$ & $m = 4n^{2c+d}$. Let $y_i = 1$ if b_i is correct (i.e., $b_i = L(x)$), otherwise $y_i = 0$. Then M' outputs incorrectly only if $Y = y_1 + \ldots + y_m \leq m/2$.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\boxed{\Pr[M(x) = L(x)] \geq ½ + |x|^{-c}}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.
- *Proof.* Let $|x| = n$ & $m = 4n^{2c+d}$. Let $y_i = 1$ if b_i is correct (i.e., $b_i = L(x)$), otherwise $y_i = 0$. Then M' outputs incorrectly only if $Y = y_1 + \ldots + y_m \leq m/2$.
- $E[y_i] = \Pr[y_i = 1] = \Pr[M(x) = L(x)] = p$ (say). It's given that $p \geq ½ + n^{-c}$. So, $\mu = E[Y] = mp \geq m/2.(1+2n^{-c})$.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.
- *Proof.* Let $|x| = n$ & $m = 4n^{2c+d}$. Let $y_i = 1$ if b_i is correct (i.e., $b_i = L(x)$), otherwise $y_i = 0$. Then M' outputs incorrectly only if $Y = y_1 + \ldots + y_m \leq m/2$.
- $E[y_i] = \Pr[y_i = 1] = \Pr[M(x) = L(x)] = p$ (say). It's given that $p \geq \frac{1}{2} + n^{-c}$. So, $\mu = E[Y] = mp \geq m/2.(1+2n^{-c})$.
- By Chernoff bound, $\Pr[Y \leq (1-\delta)\mu] \leq \exp(-(\delta^2\mu)/2)$, for any $\delta \in [0,1]$. We'll now fix the value of δ.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.
- *Proof.* $m = 4n^{2c+d}$, $p \geq \frac{1}{2} + n^{-c}$, $\mu = mp \geq m/2.(1+2n^{-c})$.
- $\Pr[Y \leq (1-\boldsymbol{\delta})\boldsymbol{\mu}] \leq \exp(-(\boldsymbol{\delta}^2\boldsymbol{\mu})/2)$, for any $\boldsymbol{\delta} \in [0,1]$.
- M' outputs incorrectly only if $Y \leq m/2$.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.
- *Proof.* $m = 4n^{2c+d}$, $p \geq \frac{1}{2} + n^{-c}$, $\boldsymbol{\mu} = mp \geq m/2.(1+2n^{-c})$.
- $\Pr[Y \leq (1-\boldsymbol{\delta})\boldsymbol{\mu}] \leq \exp(-(\boldsymbol{\delta}^2\boldsymbol{\mu})/2)$, for any $\boldsymbol{\delta} \in [0,1]$.
- M' outputs incorrectly only if $Y \leq m/2$. If we choose $\boldsymbol{\delta}$ s.t. $m/2 \leq (1-\boldsymbol{\delta})\boldsymbol{\mu}$ then $\Pr[Y < m/2] \leq \Pr[Y \leq (1-\boldsymbol{\delta})\boldsymbol{\mu}]$.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq ½ + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.
- *Proof.* $m = 4n^{2c+d}$, $p \geq ½ + n^{-c}$, $\mu = mp \geq m/2.(1+2n^{-c})$.
- $\Pr[Y \leq (1-\boldsymbol{\delta})\mu] \leq \exp(-(\boldsymbol{\delta}^2\mu)/2)$, for any $\boldsymbol{\delta} \in [0,1]$.
- M' outputs incorrectly only if $Y \leq m/2$. If we choose $\boldsymbol{\delta}$ s.t. $\boxed{m/2 \leq (1-\boldsymbol{\delta})\mu}$ then $\Pr[Y < m/2] \leq \Pr[Y \leq (1-\boldsymbol{\delta})\mu]$.
- Picking $\boldsymbol{\delta} \leq 2/(n^c+2)$ is sufficient. Set $\boldsymbol{\delta} = n^{-c}$.

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.
- *Proof.* $m = 4n^{2c+d}$, $p \geq \frac{1}{2} + n^{-c}$, $\mu = mp \geq m/2.(1+2n^{-c})$.
- $\Pr[Y \leq (1-\delta)\mu] \leq \exp(-(\delta^2\mu)/2)$, and $\delta = n^{-c}$.

- Therefore, $\Pr[M'(x) \neq L(x)] \leq \exp(-(\delta^2\mu)/2)$,

Error reduction for BPP

- Lemma. Let $c > 0$ be a constant. Suppose L is decided by a poly-time PTM M s.t. $\Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then, for every constant $d > 0$, L is decided by a poly-time PTM M' s.t. $\Pr[M'(x) = L(x)] \geq 1 - \exp(-|x|^d)$.
- *Proof.* $m = 4n^{2c+d}$, $p \geq \frac{1}{2} + n^{-c}$, $\boldsymbol{\mu} = mp \geq m/2.(1+2n^{-c})$.
- $\Pr[Y \leq (1-\boldsymbol{\delta})\boldsymbol{\mu}] \leq \exp(-(\boldsymbol{\delta}^2\boldsymbol{\mu})/2)$, and $\underline{\boldsymbol{\delta} = n^{-c}}$.
- Therefore, $\Pr[M'(x) \neq L(x)] \leq \exp(-(\boldsymbol{\delta}^2\boldsymbol{\mu})/2)$,

$$\leq \exp(-n^d).$$

Alternative definition of BPP

- Definition. A language L in BPP if there's a poly-time DTM $M(.\ ,\ .)$ and a polynomial function $q(.)$ s.t. for every $x \in \{0,1\}^*$,

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

- $2/3$ can be replaced by $1 - \exp(-|x|^d)$ as before.

(Easy Homework)

Alternative definition of BPP

- Definition. A language L in BPP if there's a poly-time DTM $M(.\ ,\ .)$ and a polynomial function $q(.)$ s.t. for every $x \in \{0,1\}^*$,

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

- Hence, $P \subseteq BPP \subseteq EXP$.

Alternative definition of BPP

- Definition. A language L in BPP if there's a poly-time <u>DTM</u> M(. , .) and a polynomial function q(.) s.t. for every $x \in \{0,1\}^*$,

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

- Hence, $P \subseteq BPP \subseteq EXP$.
- Sipser-Gacs-Lautemann. $BPP \subseteq \Sigma_2$. (We'll prove this)

Alternative definition of BPP

- Definition. A language L in BPP if there's a poly-time <u>DTM</u> M(. , .) and a polynomial function q(.) s.t. for every $x \in \{0,1\}^*$,

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

- Hence, $P \subseteq BPP \subseteq EXP$.
- Sipser-Gacs-Lautemann. $BPP \subseteq \Sigma_2$. (We'll prove this)
- How large is BPP? Is $NP \subseteq BPP$? i.e., is $SAT \in BPP$?

Alternative definition of BPP

- Definition. A language L in BPP if there's a poly-time DTM $M(. , .)$ and a polynomial function $q(.)$ s.t. for every $x \in \{0,1\}^*$,

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

- Hence, $P \subseteq BPP \subseteq EXP$.
- Sipser-Gacs-Lautemann. $BPP \subseteq \Sigma_2$. (We'll prove this)
- How large is BPP? Is $NP \subseteq BPP$? i.e., is $SAT \in BPP$?
- Next we show that $BPP \subseteq P/poly$. So, if $NP \subseteq BPP$ then $PH = \Sigma_2$. *(Karp-Lipton)*

Alternative definition of BPP

- Definition. A language L in BPP if there's a poly-time <u>DTM</u> M(. , .) and a polynomial function q(.) s.t. for every $x \in \{0,1\}^*$,

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

- Hence, $P \subseteq BPP \subseteq EXP$.
- Sipser-Gacs-Lautemann. $BPP \subseteq \Sigma_2$. (We'll prove this)

- Most complexity theorist believe that P = BPP! (More on this later.)

BPP is in P/poly

- Theorem. (*Adleman 1978*) BPP $\subseteq$ P/poly .
- *Proof.* Let $L \in$ BPP. Then, there's a poly-time <u>DTM</u> M and a polynomial function $q(.)$ s.t. for every $x \in \{0,1\}^*$,

$$\Pr_{r \in_R \{0,1\}^{q(|x|)}} [M(x, r) = L(x)] \geq 1 - 2^{-(|x|+1)} .$$

BPP is in P/poly

- Theorem. (*Adleman 1978*) BPP $\subseteq$ P/poly .
- *Proof.* Let L $\in$ BPP. Then, there's a poly-time DTM M and a polynomial function q(.) s.t. for every $x \in \{0,1\}^*$,

$$\Pr_{r \in_R \{0,1\}^{q(|x|)}} [M(x, r) = L(x)] \geq 1 - 2^{-(|x|+1)} .$$

- For every $x \in \{0,1\}^n$, at most $2^{-(n+1)}$ fraction of the r's are "bad". (r is bad for x if $M(x,r) \neq L(x)$).

BPP is in P/poly

- Theorem. (*Adleman 1978*) BPP $\subseteq$ P/poly .
- *Proof.* Let $L \in$ BPP. Then, there's a poly-time <u>DTM</u> M and a polynomial function $q(.)$ s.t. for every $x \in \{0,1\}^*$,

$$\Pr_{r \in_R \{0,1\}^{q(|x|)}} [M(x, r) = L(x)] \geq 1 - 2^{-(|x|+1)} .$$

- For every $x \in \{0,1\}^n$, at most $2^{-(n+1)}$ fraction of the r's are "bad". (r is bad for x if $M(x,r) \neq L(x)$).
- Summing over *all* $x \in \{0,1\}^n$, at most $2^n.2^{-(n+1)} = ½$ fraction of the r's are "bad" for <u>some</u> n-bit string x.

BPP is in P/poly

- Theorem. *(Adleman 1978)* $\mathrm{BPP} \subseteq \mathrm{P/poly}$.
- *Proof.* Let $L \in \mathrm{BPP}$. Then, there's a poly-time DTM M and a polynomial function $q(.)$ s.t. for every $x \in \{0,1\}^*$,

$$\Pr_{r \in_R \{0,1\}^{q(|x|)}} [M(x, r) = L(x)] \geq 1 - 2^{-(|x|+1)} .$$

- For every $x \in \{0,1\}^n$, at most $2^{-(n+1)}$ fraction of the r's are "bad". (r is bad for x if $M(x,r) \neq L(x)$).
- There's an $r_0 \in \{0,1\}^{q(n)}$ that is "good" for all $x \in \{0,1\}^n$, i.e., $M(x, r_0) = L(x)$ for all $x \in \{0,1\}^n$.

BPP is in P/poly

- Theorem. *(Adleman 1978)* BPP $\subseteq$ P/poly .
- *Proof.* Let $L \in$ BPP. Then, there's a poly-time <u>DTM</u> M and a polynomial function $q(.)$ s.t. for every $x \in \{0,1\}^*$,

$$\Pr_{r \in_R \{0,1\}^{q(|x|)}} [M(x, r) = L(x)] \geq 1 - 2^{-(|x|+1)} .$$

- For every $x \in \{0,1\}^n$, at most $2^{-(n+1)}$ fraction of the r's are "bad". (r is bad for x if $M(x,r) \neq L(x)$).
- There's an $r_0 \in \{0,1\}^{q(n)}$ that is "good" for all $x \in \{0,1\}^n$, i.e., $M(x, r_0) = L(x)$ for all $x \in \{0,1\}^n$.
- By hardwiring this r_0, the computation of $M(. , r_0)$ can be viewed as a poly(n)-size circuit C. *(Cook-Levin)* ∎

Why truly random bits?

- A PTM is defined using truly random bits. Is the definition sufficiently powerful? Do <u>*biased*</u> random bits give any additional computational power?

Why truly random bits?

- A PTM is defined using truly random bits. Is the definition sufficiently powerful? Do *biased* random bits give any additional computational power?

- Claim. A random bit with $Pr[1] = p$ can be simulated by a PTM in expected $O(1)$ time if the i-th bit of p can be computed in $poly(i)$ time. *(Homework)*

Why truly random bits?

- A PTM is defined using truly random bits. Is the definition sufficiently powerful? Do *biased* random bits give any additional computational power?

- Claim. A random bit with $\Pr[1] = p$ can be simulated by a PTM in expected $O(1)$ time if the i-th bit of p can be computed in $poly(i)$ time. *(Homework)*

- There's a p and a PTM M with access to p-biased random bits s.t. M decides an undecidable language!

Why truly random bits?

- On the other hand, we can obtain truly random bits from biased random bits.

- Claim. *(von-Neumann 1951)* A truly random bit can be simulated by a PTM with access to p-biased random bits in expected $O(p^{-1}(1-p)^{-1})$ time. *(Homework)*