Complexity and Cryptography

Thomas Zeugmann

Hokkaido University
Laboratory for Algorithmics

https://www-alg.ist.hokudai.ac.jp/~thomas/COCRB/

Lecture 10: Probabilistic Complexity Classes

Probabilistic Turing Machines I

After having studied deterministic and nondeterministic complexity classes, we finish our study of complexity by taking a closer look at probabilistic complexity classes.

Probabilistic Turing Machines I

After having studied deterministic and nondeterministic complexity classes, we finish our study of complexity by taking a closer look at probabilistic complexity classes.

First, we have to define probabilistic Turing machines. This is done informally.

Probabilistic Turing machines (abbr. PTM) are defined as deterministic Turing machines except that they have an additional tape equipped with a one-way read-only head. On this auxiliary tape, an infinite sequence of zeros and ones is written. These zeros and ones are realizations of a sequence of coin flips. It is assumed that zero and one each have probability 1/2. Moreover, the coin flips are independent of one another. Furthermore, it assumed that each realization; i.e., each infinite sequence of zeros and ones written on the auxiliary tape is independent of all other such sequences.

Probabilistic Turing Machines II

On input a string w, the PTM works as a deterministic Turing machine. In each step of its computation it may read one symbol from the auxiliary tape. If it does, the one-way read-only head of the auxiliary tape moves one position to the right. Consequently, a PTM may obtain different results on the same input, but the result is determined for every random sequence written on the auxiliary tape.

Probabilistic Turing Machines II

On input a string w, the PTM works as a deterministic Turing machine. In each step of its computation it may read one symbol from the auxiliary tape. If it does, the one-way read-only head of the auxiliary tape moves one position to the right. Consequently, a PTM may obtain different results on the same input, but the result is determined for every random sequence written on the auxiliary tape.

So, it remains to redefine the notion of acceptance. We present here the usual definition found in the literature.

Probabilistic Turing Machines III

Definition 1

Let p be a constant such that $1/2 < p \leqslant 1$, let P be a PTM and let $w \in \Sigma^*$. The PTM P is said to *accept* w provided the probability of the following event E is greater than or equal to p:

Event E. P stops in an accepting configuration.

Moreover, we write $L_p(P)$ to denote the set of all strings $w \in \Sigma^*$ that are accepted by P with probability greater than or equal to p.

Probabilistic Turing Machines IV

The space and time complexity of a PTM P are defined by requesting P to obey a space or time bound $f(n)$ in the event E defined above. That is, on all accepting computations P uses at most $f(n)$ cells on all its work tapes (for space complexity) and/or works at most $f(n)$ steps (for time complexity).

Probabilistic Turing Machines IV

The space and time complexity of a PTM P are defined by requesting P to obey a space or time bound $f(n)$ in the event E defined above. That is, on all accepting computations P uses at most $f(n)$ cells on all its work tapes (for space complexity) and/or works at most $f(n)$ steps (for time complexity).

We are not going to provide a formal proof for the correctness of our definition, since this is beyond the scope of this course.

Probabilistic Turing Machines IV

The space and time complexity of a PTM P are defined by requesting P to obey a space or time bound $f(n)$ in the event E defined above. That is, on all accepting computations P uses at most $f(n)$ cells on all its work tapes (for space complexity) and/or works at most $f(n)$ steps (for time complexity).

We are not going to provide a formal proof for the correctness of our definition, since this is beyond the scope of this course.

Before discussing further issues of PTMs, we exemplify their power by looking again at the language

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}.$$

The following theorem was found by Rūsiņš Freivalds:

Probabilistic Turing Machines V

Theorem 1 (Freivalds (1981))

For every p *with* $1/2 < p < 1$ *there exists a constant* $\kappa > 0$ *and a PTM* P *accepting* L *with probability* p *which uses only* κ *many cells on all its work tapes.*

Probabilistic Turing Machines V

Theorem 1 (Freivalds (1981))

For every p with $1/2 < p < 1$ there exists a constant $\kappa > 0$ and a PTM P accepting L with probability p which uses only κ many cells on all its work tapes.

Proof. Let p with $1/2 < p < 1$ be given. First, we choose two numbers $c,\ d \in \mathbb{N}$; $c,\ d > 1$, depending on p such that

$$1 - 2 \cdot \left(\frac{1}{2}\right)^d \geqslant p \quad \text{and} \tag{1}$$

$$\left(\frac{2^c}{2^c + 1}\right)^d > 1 - p \tag{2}$$

are satisfied.

Probabilistic Turing Machines VI

We define the desired PTM P (again depending on p). On input $w \in \{0,1\}^*$ the PTM P does the following:

(1) The PTM P checks whether or not the input w has the form 0^m1^n for some $m, n \in \mathbb{N}$. If this is not the case, P rejects w. Otherwise, P continues by executing (2).

(2) The PTM P checks whether or not $(m - n) \mod c = 0$. If this is not the case, P rejects w.
Otherwise, P continues by executing (3) (see below).

Probabilistic Turing Machines VI

We define the desired PTM P (again depending on p). On input $w \in \{0,1\}^*$ the PTM P does the following:

(1) The PTM P checks whether or not the input w has the form 0^m1^n for some $m, n \in \mathbb{N}$. If this is not the case, P rejects w. Otherwise, P continues by executing (2).

(2) The PTM P checks whether or not $(m - n) \mod c = 0$. If this is not the case, P rejects w.
Otherwise, P continues by executing (3) (see below).

Note that P, while executing (1) and (2), does not read any symbol on its auxiliary tape. Clearly, (1) can be performed without using any cells on the work tapes. It suffices to scan the input ones, and then to return the head of the input tape to the first position of the input.

Probabilistic Turing Machines VII

For executing (2), P does not use any cell of its work tapes.
It suffices to count the number of zeros modulo c (by using appropriate states), and then the number of ones, where P memorizes the outcome of counting the zeros modulo c in an appropriate state s_r. Here r stands for the remainder, i.e., for $m \mod c$. From s_r, the PTM P can switch to c different states (all memorizing s_r). If $n \bmod c = r$, then P switches to the particular state in which it can start executing (3).
Otherwise, the input is already rejected, and P stops.

Probabilistic Turing Machines VII

For executing (2), P does not use any cell of its work tapes.
It suffices to count the number of zeros modulo c (by using appropriate states), and then the number of ones, where P memorizes the outcome of counting the zeros modulo c in an appropriate state s_r. Here r stands for the remainder, i.e., for $m \mod c$. From s_r, the PTM P can switch to c different states (all memorizing s_r). If $n \bmod c = r$, then P switches to the particular state in which it can start executing (3).
Otherwise, the input is already rejected, and P stops.

Now, P has to figure out whether or not $m = n$. This is done by performing the following *probabilistic experiment*:

Probabilistic Turing Machines VIII

(3) P starts on the leftmost symbol of the input and reads the whole input from left to right, thereby also reading in each step a symbol from its auxiliary tape. This is done until the last input symbol has been read. We refer to this procedure as to a *run* of the experiment.
We refer to the string 0^m as the left block and to 1^n as the right block of the input.
A *run* of the experiment is *successful for the left block* if P reads only zeros on its auxiliary tape while reading the zeros on its input tape.
A run of the experiment is *successful for the right block* if P reads only ones on its auxiliary tape while reading the ones on its input tape.

Probabilistic Turing Machines IX

(3) A *set* of the experiment consists of successive runs. A set is finished if either the run for the left block of the input was successful or the run for the right block of the input was successful. The set is then said to be won for the respective block.
The experiment consists of d sets.
The PTM P accepts the input w iff each block could win at least one set.

Probabilistic Turing Machines X

When starting (3), there are only two cases left, i.e., $m = n$ or $|m - n| \geqslant c$. It is clear that for the execution of the experiment only a constant number κ of tape cells is sufficient.

Probabilistic Turing Machines X

When starting (3), there are only two cases left, i.e., $m = n$ or $|m - n| \geqslant c$. It is clear that for the execution of the experiment only a constant number κ of tape cells is sufficient.

Also, it is clear that the probability to arrive at a run successful for either the left of right block, say ε, satisfies $\varepsilon > 0$. Hence, $1 - \varepsilon < 1$ is the probability that a run was neither successful for the left block nor for the right block or it was successful for both blocks. Since $\lim_{n\to\infty}(1-\varepsilon)^n = 0$, we directly see that the probability to finish a set tends to 1 if the number of steps performed by P tends to infinity.

Probabilistic Turing Machines XI

Consequently, it remains to calculate the probability of acceptance for the input for both cases.

Probabilistic Turing Machines XI

Consequently, it remains to calculate the probability of acceptance for the input for both cases.

Case 1. $n = m$.

In this case, the probability to win is for each block the same, i.e., $1/2$. Therefore, the probability for the left block to win all d sets is $(1/2)^d$ and so is the probability for the right block to win all d sets. Thus, the probability that one of the blocks wins all d sets is $2 \cdot (1/2)^d$. By Inequality (1) we can conclude

$$1 - 2 \cdot \left(\frac{1}{2}\right)^d \geqslant p \,,$$

and thus, with probability greater than or equal to p the input is accepted.

Probabilistic Turing Machines XII

Case 2. $|m - n| \geqslant c$.

Without loss of generality, $m < n$, and thus, $m + c \leqslant n$.

Now, the conditional probability for the left block to win is

$$\frac{2^{-m}(1-2^{-n})}{2^{-m}(1-2^{-n})+2^{-n}(1-2^{-m})} > \frac{2^{-m}}{2^{-n}+2^{-m}}$$

$$= \frac{2^c}{2^c+2^{-n+m+c}} \geqslant \frac{2^c}{2^c+1}.$$

Probabilistic Turing Machines XII

Case 2. $|m - n| \geqslant c$.

Without loss of generality, $m < n$, and thus, $m + c \leqslant n$.

Now, the conditional probability for the left block to win is

$$\frac{2^{-m}(1-2^{-n})}{2^{-m}(1-2^{-n})+2^{-n}(1-2^{-m})} > \frac{2^{-m}}{2^{-n}+2^{-m}}$$

$$= \frac{2^c}{2^c+2^{-n+m+c}} \geqslant \frac{2^c}{2^c+1}.$$

Consequently, the probability for the left block to win all d sets can be lower bounded by

$$\left(\frac{2^c}{2^c+1}\right)^d.$$

Probabilistic Turing Machines XIII

Finally, by Inequality (2) we can conclude

$$\left(\frac{2^c}{2^c+1}\right)^d > 1-p \quad \text{and, therefore}$$

$$\begin{aligned} 1-\left(\frac{2^c}{2^c+1}\right)^d &< 1-(1-p) \\ &= p\,. \end{aligned}$$

Probabilistic Turing Machines XIII

Finally, by Inequality (2) we can conclude

$$\left(\frac{2^c}{2^c+1}\right)^d > 1-p \quad \text{and, therefore}$$

$$\begin{aligned} 1-\left(\frac{2^c}{2^c+1}\right)^d &< 1-(1-p) \\ &= p\,. \end{aligned}$$

That is, the probability to accept a string of the form 0^m1^n with $m < n$; i.e., not belonging to L, is *less* than p. ∎

Probabilistic Turing Machines XIV

So, we have just seen what amazing computational power probabilism can provide if the computational resources are severely restricted. But some remarks are in order here.

Probabilistic Turing Machines XIV

So, we have just seen what amazing computational power probabilism can provide if the computational resources are severely restricted. But some remarks are in order here.

The PTM presented in the proof above is very space efficient, but not time efficient. Performing the probabilistic experiment described takes a huge amount of time.

Probabilistic Turing Machines XIV

So, we have just seen what amazing computational power probabilism can provide if the computational resources are severely restricted. But some remarks are in order here.

The PTM presented in the proof above is very space efficient, but not time efficient. Performing the probabilistic experiment described takes a huge amount of time.

In the following we are mainly interested in algorithms that have an efficient run-time. Thus, we shall restrict ourselves to consider PTMs obeying an (expected) polynomial bound for their run time. This line of research was initiated by John Gill.

The Complexity Class $\mathcal{PP}$

Definition 2 (Gill (1977))

The *probabilistic polynomial* ($\mathcal{PP}$) class is the set of languages L for which there is a PTM P running in polynomial-time such that for all strings x we have

(a) $x \in L \Longrightarrow \Pr(x \text{ is accepted}) > 1/2$,

(b) $x \notin L \Longrightarrow \Pr(x \text{ is rejected}) > 1/2$.

The Complexity Class $\mathcal{PP}$

Definition 2 (Gill (1977))

The *probabilistic polynomial* ($\mathcal{PP}$) class is the set of languages L for which there is a PTM P running in polynomial-time such that for all strings x we have

(a) $x \in L \Longrightarrow \Pr(x \text{ is accepted}) > 1/2$,

(b) $x \notin L \Longrightarrow \Pr(x \text{ is rejected}) > 1/2$.

It can be said that $\mathcal{PP}$ is the weakest class of problems that can be approximately *solved* in the very intuitive sense of the word. Moreover, it seems that any problem not in $\mathcal{PP}$ will take more than polynomial-time to be solved. However, no proof is known.

Remark

Requiring the majority of answers to be correct is a natural idea. But if the correct solution is given with a probability close to 1/2 it is hard to differentiate the incorrect solution from the correct one.

Remark

Requiring the majority of answers to be correct is a natural idea. But if the correct solution is given with a probability close to 1/2 it is hard to differentiate the incorrect solution from the correct one.

Moreover, as the input size grows, we may be faced with the problem that the probability of the correct solution tends more and more to 1/2. Thus, such a machine will be hard to distinguish from a machine that is simply guessing without performing any computation. This gives way to a more restrictive definition which we present next. Now, we are going to bound away from 1/2 the margin of the solution.

The Complexity Class $\mathcal{BPP}$

Definition 3 (Gill (1977))

The *bounded probabilistic polynomial* ($\mathcal{BPP}$) class is the set of languages L for which there is a PTM P running in polynomial-time and some constant $\varepsilon > 0$ such that for all strings x we have

(a) $x \in L \Longrightarrow \Pr(x \text{ is accepted}) > 1/2 + \varepsilon$,

(b) $x \notin L \Longrightarrow \Pr(x \text{ is rejected}) > 1/2 + \varepsilon$.

The Complexity Class $\mathcal{BPP}$

Definition 3 (Gill (1977))

The *bounded probabilistic polynomial* ($\mathcal{BPP}$) class is the set of languages L for which there is a PTM P running in polynomial-time and some constant $\varepsilon > 0$ such that for all strings x we have

(a) $x \in L \Longrightarrow \Pr(x \text{ is accepted}) > 1/2 + \varepsilon$,

(b) $x \notin L \Longrightarrow \Pr(x \text{ is rejected}) > 1/2 + \varepsilon$.

We get $\mathcal{BPP} \subseteq \mathcal{PP}$, since $1/2 + \varepsilon > 1/2$ for all $\varepsilon > 0$. Note that Definition 3 can be modified and still gives the same complexity class. In particular, for any constant $\varepsilon \in (0, 1/2)$ we again arrive at $\mathcal{BPP}$.

The class $\mathcal{BPP}$ contains all languages that can be accepted by efficient Monte-Carlo algorithms.

The Complexity Class $\mathcal{RP}$

Moreover, we can further sharpen the definition of $\mathcal{BPP}$ by requiring that the PTM is making only one type of error.

More precisely, we require that for all inputs $x \notin L$, the PTM for accepting L makes *no* error. Furthermore, every string from L must be accepted with probability at least 1/2. Again, we could replace 1/2 by any constant greater than 1/2 and less than 1. The resulting complexity class is denoted by $\mathcal{RP}$. Intuitively, $\mathcal{RP}$ stands for *random polynomial time* though this term may be a bit misleading.

More formally, we arrive at the following definition:

The Complexity Class $\mathcal{RP}$

Definition 4 (Gill (1977))

The *one-sided error probabilistic polynomial* ($\mathcal{RP}$) class is the set of languages L for which there is a PTM P running in polynomial-time such that for all strings x we have

(a) $x \in L \Longrightarrow \Pr(x \text{ is accepted}) > 1/2$,

(b) $x \notin L \Longrightarrow x$ is rejected.

The Complexity Class $\mathcal{RP}$

Definition 4 (Gill (1977))

The *one-sided error probabilistic polynomial* ($\mathcal{RP}$) class is the set of languages L for which there is a PTM P running in polynomial-time such that for all strings x we have

(a) $x \in L \Longrightarrow \Pr(x \text{ is accepted}) > 1/2$,

(b) $x \notin L \Longrightarrow x$ is rejected.

As a matter of fact, we already know an important problem belonging to $\mathcal{RP}$. Recalling our results obtained in Lecture 5, we have seen that $\overline{PRIM} \in \mathcal{RP}$, where $PRIM$ denotes the set of all binary representations of prime numbers.

The Complexity Class $\mathcal{ZPP}$

The last probabilistic complexity class we are going to define represents the efficient Las Vegas algorithms.

Here, *no error* whatsoever is allowed and the expected run time must be uniformly bounded by a polynomial in the length of all inputs over the underlying alphabet Σ.
This class is denoted by $\mathcal{ZPP}$ and it was also defined by Gill (1977).
So, $\mathcal{ZPP}$ stands for "zero-error probabilistic polynomial time."

The Complexity Class $\mathcal{ZPP}$

The last probabilistic complexity class we are going to define represents the efficient Las Vegas algorithms.

Here, *no error* whatsoever is allowed and the expected run time must be uniformly bounded by a polynomial in the length of all inputs over the underlying alphabet Σ.
This class is denoted by $\mathcal{ZPP}$ and it was also defined by Gill (1977).
So, $\mathcal{ZPP}$ stands for "zero-error probabilistic polynomial time."

Theorem 2 establishes the more obvious relations with respect to set inclusion between the probabilistic complexity complexity classes and some other previously defined complexity classes.

Results I

Let us recall the notion of a balanced TM.

Definition 5

An NTM is *balanced* if all computation paths over a string x are of the same length and, moreover, each state is a guess state.

Results I

Let us recall the notion of a balanced TM.

Definition 5

An NTM is *balanced* if all computation paths over a string x are of the same length and, moreover, each state is a guess state.

Now, it is easy to see that every $\mathcal{NP}$ machine M can be replaced by a balanced NTM M' such that $L(M) = L(M')$.

Results II

Furthermore, the notion of a balanced NTM can be easily adapted to a PTM. The only modification to be made is to replace “guess state” by “coin-tossing” state.

Results II

Furthermore, the notion of a balanced NTM can be easily adapted to a PTM. The only modification to be made is to replace "guess state" by "coin-tossing" state.

Exercise 1. *Prove that for every PTM* P *accepting a language* L *in the sense of* $\mathcal{PP}$ *there is balanced PTM* P' *such that* $L(P) = L(P') = L$.

Results III

Theorem 2

(1) $\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP} \subseteq \mathcal{BPP} \subseteq \mathcal{PP} \subseteq \mathcal{PSPACE}$;

(2) $\mathcal{RP} \subseteq \mathcal{NP} \subseteq \mathcal{PP}$;

(3) *$\mathcal{ZPP}$, $\mathcal{RP}$ and $\mathcal{BPP}$ are closed under union and intersection.*

Results III

Theorem 2

(1) $\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP} \subseteq \mathcal{BPP} \subseteq \mathcal{PP} \subseteq \mathcal{PSPACE}$;

(2) $\mathcal{RP} \subseteq \mathcal{NP} \subseteq \mathcal{PP}$;

(3) *$\mathcal{ZPP}$, $\mathcal{RP}$ and $\mathcal{BPP}$ are closed under union and intersection.*

First, we prove Assertion (1). Clearly, by definition we have $\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP}$ and $\mathcal{BPP} \subseteq \mathcal{PP}$ (as already mentioned above).

Results III

Theorem 2

(1) $\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP} \subseteq \mathcal{BPP} \subseteq \mathcal{PP} \subseteq \mathcal{PSPACE}$;

(2) $\mathcal{RP} \subseteq \mathcal{NP} \subseteq \mathcal{PP}$;

(3) *$\mathcal{ZPP}$, $\mathcal{RP}$ and $\mathcal{BPP}$ are closed under union and intersection.*

First, we prove Assertion (1). Clearly, by definition we have $\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP}$ and $\mathcal{BPP} \subseteq \mathcal{PP}$ (as already mentioned above).

For seeing that $\mathcal{PP} \subseteq \mathcal{PSPACE}$ it suffices to notice that a PTM P can be simulated by a deterministic TM M which performs all possible computations of P. Additionally, M counts the number of accepting computations and makes at the end a majority vote. We omit the details.

Results IV

Claim: $\mathcal{RP} \subseteq \mathcal{BPP}$

Let L be accepted in the sense of $\mathcal{RP}$ by a PTM P. Thus, if $x \notin L$ then x is always rejected.

If $x \in L$, then $\Pr(x \text{ is accepted}) > 1/2$. Thus, we can construct a PTM P' which behaves as follows: On every input x it runs the PTM P exactly twice. The PTM P' accepts x, if x has been accepted by P at least once. Hence, if $x \notin L$, then P' will never accept x, too. On the other hand, if $x \in L$, then P' makes an error if and only if P has made an error on x twice. The probability that P is not accepting x twice is, however, less than 1/4. Hence P' accepts L in the sense of $\mathcal{BPP}$. This proves Assertion (1).

$\mathcal{RP} \subseteq \mathcal{NP}$

Assume a language L that is accepted in the sense of $\mathcal{RP}$ by a PTM P. Again, if $x \notin L$, then x is always rejected. Thus, we can simply remove the coin-flips made by P and replace them by a nondeterministic choice. Hence, the resulting machine M nondeterministically accepts L. This proves $\mathcal{RP} \subseteq \mathcal{NP}$.

$\mathcal{NP} \subseteq \mathcal{PP}$

Next, we have to show that $\mathcal{NP} \subseteq \mathcal{PP}$. Let $L \in \mathcal{NP}$ be witnessed by the NTM M.
First recall that for all $x \notin L$ there is no accepting computation of M for x.
But if $x \in L$, then at least one accepting computation of M for x must exist. However, the desired PTM P has to accept/reject a string x if the majority of the computations performed is accepting/rejecting x. For reaching this goal, we proceed as follows:

$\mathcal{NP} \subseteq \mathcal{PP}$

Next, we have to show that $\mathcal{NP} \subseteq \mathcal{PP}$. Let $L \in \mathcal{NP}$ be witnessed by the NTM M.
First recall that for all $x \notin L$ there is no accepting computation of M for x.
But if $x \in L$, then at least one accepting computation of M for x must exist. However, the desired PTM P has to accept/reject a string x if the majority of the computations performed is accepting/rejecting x. For reaching this goal, we proceed as follows:
Let M be a balanced NTM and let $L(M)$ be the language accepted by M. Moreover, there exists a polynomial p such that M takes time $p(|x|)$ on all inputs x. Without loss generality we can also assume that $\Sigma = \{0, 1\}$. We construct a balanced PTM P from the machine M as follows:

$\mathcal{NP} \subseteq \mathcal{PP}$

First, each guessing state is *replaced* by a coin-tossing state. The outcome of a coin-toss then corresponds to a guess. Each terminal node of the computation tree of M over a string x with $n = |x|$ is then reached with probability $1/2^{p(n)}$.

Now, a further computation is carried out at each such leaf. That is, the terminal leaves of M's computation tree on x are no longer terminal leaves for the computation tree of P over x.

$\mathcal{NP} \subseteq \mathcal{PP}$

Let q be a polynomial such that $p(n) \leqslant q(n)$ for all $n \in \mathbb{N}$.
If the terminal leaf of M's computation tree on x is rejecting, then P will toss its coin $q(n)$ times. If not all outcomes of these coin tosses are head, then P tosses the coin again.
It accepts if the last coin toss is head and rejects x otherwise.
If all outcomes of these $q(n)$ coin tosses are head, then P tosses the coin again (for being balanced) but rejects the input x regardless of the outcome of the last coin toss.

$\mathcal{NP} \subseteq \mathcal{PP}$

Let q be a polynomial such that $p(n) \leqslant q(n)$ for all $n \in \mathbb{N}$.
If the terminal leaf of M's computation tree on x is rejecting, then P will toss its coin $q(n)$ times. If not all outcomes of these coin tosses are head, then P tosses the coin again.
It accepts if the last coin toss is head and rejects x otherwise.
If all outcomes of these $q(n)$ coin tosses are head, then P tosses the coin again (for being balanced) but rejects the input x regardless of the outcome of the last coin toss.

On the other hand, if the reached terminal leaf in M's computation tree on x is accepting, then machine P tosses its coin again $q(n) + 1$ times and accepts regardless of the outcome (again this done only to make P balanced).

$\mathcal{NP} \subseteq \mathcal{PP}$

Case 1. $x \notin L(M)$.

Then, the definition of P directly implies that

$$\begin{aligned} \Pr(x \text{ is rejected} | x \notin L) &= \frac{2^{p(n)}\left(2^{q(n)}+1\right)}{2^{p(n)+q(n)+1}} \\ &= \frac{1}{2} + \frac{1}{2^{q(n)+1}} . \end{aligned}$$

$\mathcal{NP} \subseteq \mathcal{PP}$

Case 2. $x \in L(M)$.

$$\begin{aligned}
\Pr(x \text{ is acc.} \mid x \in L) &\geqslant \frac{\left(2^{p(n)}-1\right)\left(2^{q(n)}-1\right)}{2^{p(n)+q(n)+1}} + \frac{2^{q(n)+1}}{2^{p(n)+q(n)+1}} \\
&= \frac{1}{2} + \frac{2^{q(n)} - 2^{p(n)} + 1}{2^{p(n)+q(n)+1}} \\
&> \frac{1}{2} \quad \text{since } q(n) \geqslant p(n)\,.
\end{aligned}$$

Hence P accepts $L(M)$ in the sense of $\mathcal{PP}$. This completes the proof of Assertion (2).

$\mathcal{ZPP}$ is closed under union and intersection

Consider two PTMs P and P' such that $L(P)$ and $L(P')$ are accepted in the sense of $\mathcal{ZPP}$.
Then it easy to see that a PTM $\hat{P}$ accepts $L(P) \cap L(P')$
($L(P) \cup L(P')$) in the sense of $\mathcal{ZPP}$ if $\hat{P}$ works as follows:
It simulates both P and P' and accepts its input x if P and P' accept x (if P or P' accept x).

We leave it as an exercise to show the closure with respect to union and disjunction for the remaining classes mentioned in Assertion (3). ∎

$\mathcal{ZPP} = \mathcal{RP} \cap \text{co-}\mathcal{RP}$

Next, we mention the following characterization for $\mathcal{ZPP}$:

Theorem 3

$\mathcal{ZPP} = \mathcal{RP} \cap \text{co-}\mathcal{RP}$.

The proof is provided in the book.

Final Remarks

We like to conclude our short excursion into the field of randomized computations by mentioning that there is much more.
In particular, it is not too hard to prove that there are complete problems for $\mathcal{PP}$. The perhaps easiest $\mathcal{PP}$-complete problem is *MAJ* defined as the set of all Boolean formulae that are satisfied by the majority of possible assignments of the variables occurring in them.

Final Remarks

We like to conclude our short excursion into the field of randomized computations by mentioning that there is much more.
In particular, it is not too hard to prove that there are complete problems for $\mathcal{PP}$. The perhaps easiest $\mathcal{PP}$-complete problem is *MAJ* defined as the set of all Boolean formulae that are satisfied by the majority of possible assignments of the variables occurring in them.

Furthermore, it has been a long standing open problem whether or not $\mathcal{PP}$ is also closed under union and intersection. This problem got solved in 1991, but the proof technique used is too complex to be included here. We refer the reader to Beigel *et al.* (1991).

Final Remarks

But we do *not* know any problem that is complete for $\mathcal{BPP}$ under polynomial time reductions.

Final Remarks

But we do *not* know any problem that is complete for $\mathcal{BPP}$ under polynomial time reductions.

One reason for the difficulty to find a problem that is complete for $\mathcal{BPP}$ is that the defining property of the class $\mathcal{BPP}$ is *semantic*. That is, for *every* string x over the underlying alphabet, a Turing machine has either to accept x with probability at least $1/2 + \varepsilon$ or it has to reject it with probability at least $1/2 + \varepsilon$. Given a description of a Turing machine, it is undecidable whether or not it has this property.

Summary

The following figure summarizes the known inclusions between the probabilistic complexity classes and relates them to $\mathcal{P}$ and $\mathcal{PSPACE}$, where A $\rightarrow$ B stands for A $\subseteq$ B:

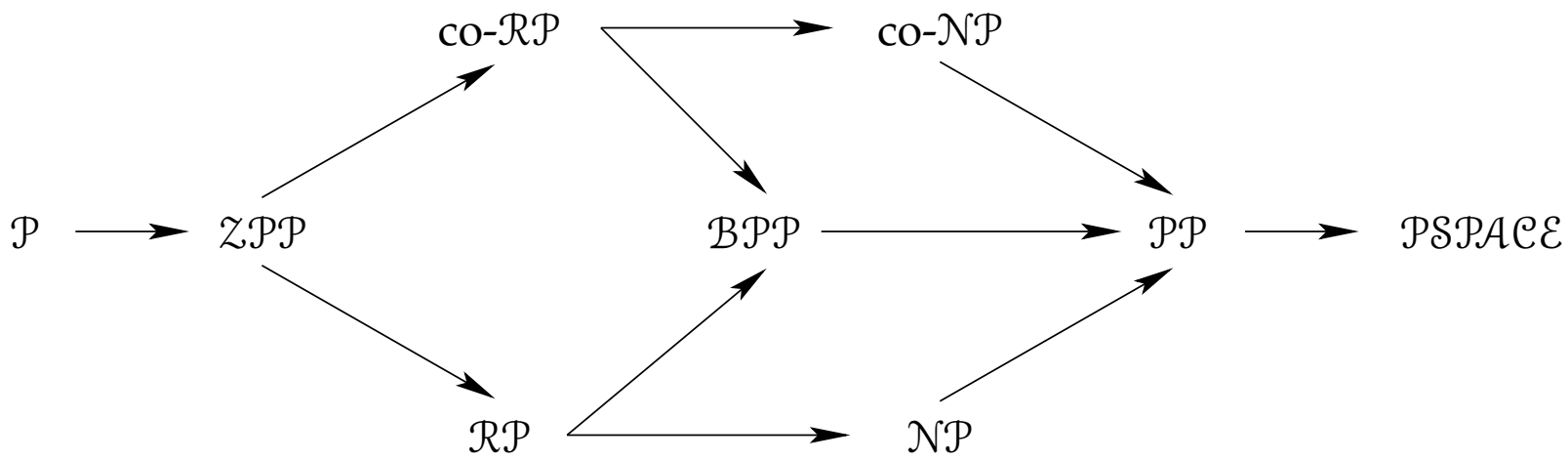

Figure 1: Inclusions between complexity classes.

Thank you!

Rūsiņš Freivalds

John Gill