Journal of Universal Computer Science, vol. 12, no. 6 (2006), 746-761
submitted: 31/5/06, accepted: 23/6/06, appeared: 28/6/06 © *J.UCS*

Randomized Algorithms and Complexity Theory

Harald Hempel
(Friedrich-Schiller-Universität Jena, Germany
hempel@informatik.uni-jena.de)

Abstract: In this paper we give an introduction to the connection between complexity theory and the study of randomized algorithms. In particular, we will define and study probabilistic complexity classes, survey the basic results, and show how they relate to the notion of randomized algorithms.
Key Words: Randomized algorithms, randomized complexity classes
Category: F.1.2, F.1.3, F.2.2, F.2.3

1 Introduction

Truly good algorithms solve problems fast and deliver optimal solutions. Unfortunately, many real world problems do not seem to have such good algorithms. Consequently, algorithm designers have to settle for less, algorithms that still deliver optimal solutions but take way too much time as the size of the input grows, or algorithms that run fast but obtain only approximations of the optimal solution. Beside those two, there seems to be a third type of algorithm that has turned out to be quite useful for some applications: randomized algorithms. Randomized algorithms are algorithms in which some computational steps depend on random choices and so the outcome itself is a random variable. However, randomized algorithms are fast and the prize one has to pay for that is that the solution they compute is correct only with a certain probability. Good randomized algorithms are those that come with a very high probability that the result they compute is correct.

We will study randomized algorithms in the setting of complexity theory, in particular we will focus on probabilistic complexity classes. Complexity theory mainly deals with the issue what amount of time and space is needed to solve a problem, the so-called complexity of a problem. Problems with similar time and space bounds are grouped in complexity classes and the structure and properties of such complexity classes are studied.

When we speak of problem we usually refer to decision problems, problems that have YES/NO-solutions. Even though many real world problems are not of that decision nature but rather of the nature that a function value is asked, the difference between the complexity of those two is usually quite small. In fact, computing a single bit of a function is just a decision problem and so by repeatedly solving a decision problem one can easily compute the value of a

function. So in the following we will restrict ourself to the study of decision problems, in other words the acceptance of sets of strings or languages.

The model of computation used in complexity theory to analyze the amount of time and space needed to solve a problem is the Turing machine. The notion of a nondeterministic Turing machine (NTM) can be used as a model for randomized algorithms and so will be important for this survey. A formal definition of the concept of a nondeterministic Turing machine can be found in any standard textbook on complexity theory (for instance [Pap94]). Informally put, a nondeterministic Turing machine is an algorithm which in each computation step has two choices[1] for continuing the computation. In contrast to the deterministic Turing machine where there is just one outcome of the computation, and so acceptance of string by that algorithm is defined in the obvious way, a nondeterministic Turing machine may have many "computation paths," and each of those is potentially able to reject or accept the input independent of what other computation paths do. We therefore say a nondeterministic Turing machine accepts an input if and only if there exists a sequence of choices during the run of the algorithm such that the input is accepted on this computation path. An NTM M accepts a language L if and only if for all strings $x \in \Sigma^*$ the string x is in L if and only if x is accepted by L. The set of all languages accepted by NTMs whose runtime is bounded by some polynomial defines the well-known complexity class NP.

However, the above acceptance criteria is not the only reasonable acceptance criteria for nondeterministic computations. Basing acceptance or rejection of the input on the outcome of the majority of the computation paths leads us to so-called probabilistic complexity classes, the complexity-theoretic model of randomized algorithms. A so-called probabilistic Turing machine is just a nondeterministic Turing machine in which whenever there are two possible ways of continuing the computation, a fair coin flip is made to govern the computation. From that perspective the class NP is the set of all languages L such that there is a probabilistic Turing machine M such that the following holds for each $x \in \Sigma^*$:

$$x \in L \iff Prob(M \text{ accepts } x) > 0.$$

Note that this type of randomized algorithm has a nonzero success probability but the probability that on a single run an input is falsely rejected can be very close to 1. On the other hand, accepting strings on a single run that do not belong to the language is impossible. So NP can be viewed as the class of languages that have randomized algorithms with one-sided nonzero success probability. Clearly, one would usually not call such NP-algorithms randomized algorithms, but for the scope of this paper we will do so. We will soon define classes of problems that admit "true" randomized algorithms.

[1] In general, nondeterminism allows more than just two choices but restricting the model to only two can be done without loss of generality.

The paper is organized as follows. In Section 2 we will define the basic notions and probabilistic complexity classes. Section 3 contains an overview over the basic containment relations between the probabilistic complexity classes. In Sections 4, 5, 6, 7, and 8 we will study each of the probabilistic classes NP, PP, BPP, RP, and ZPP in more detail. Finally, Section 9 will contain historical remarks and literature pointers.

2 Preliminaries

The decision problems we consider are sets of strings over the alphabet $\Sigma = \{0, 1\}$. As already mentioned in Section 1, our model of computation is a variant of the nondeterministic Turing machine, for a detailed definition see for instance [Pap94]. Interpreting the nondeterministic choices of such a machine as random events we obtain so-called probabilistic Turing machines that allow to model randomized algorithms. For clarity of presentation, we will work without loss of generality with the following assumptions regarding probabilistic Turing machines:

1. The runtime depends solely on the length of the input. To be more precise, there is a polynomial p such that on any input $x \in \Sigma^*$ and for any sequence of coin flips during the run of the Turing machine on input x, the machine makes exactly $p(|x|)$ computation steps.

2. The number of coin flips made during the run of the machine on an input $x \in \Sigma^*$ depends solely on $|x|$, and hence for any sequence of coin flips during the run of the machine on input x, the number of coin flips made is the same. (We can even assume that there is a coin flip made in every computation step.)

3. We assume fair coin flips and so a uniform distribution for the series of coin flips, i.e., each single coin flip is independent of any previous coin flip and returns 1 (heads) or 0 (tails) with probability exactly $\frac{1}{2}$.

4. The Turing machine, for any input and for any series of coin flips made during the run on that input, always ends in one of two possible final states: Accept or Reject.

Randomized algorithms in general have the property that a single run does not give a reliable answer whether the input is truly accepted or not. In fact, a randomized algorithm accepts the correct and rejects the incorrect inputs only with a certain probability. For a probabilistic Turing machine M, a string $x \in \Sigma^*$, and a series of coin flips y, without loss of generality assume $y \in \{0, 1\}^{p(|x|)}$, where p is a polynomial, and let $M(x, y)$ denote the final state reached by M on input x if the computation proceeded according to the series of coin flips encoded

by y (each bit of y encodes one coin flip). We write $M(x,y) = +$ if the final state reached on input x and the computation proceeding according to the series of coin flips y is "Accept," and we write $M(x,y) = -$ if the final state is "Reject." Observe that since y is a series of random events, also $M(x,y)$ is a random event. The probability of a Turing machine M to reach the "Accept" or "Reject" state on input x will be denoted by $Prob\{y : M(x,y) = +\}$ or $Prob\{y : M(x,y) = -\}$ and by $Prob(M(x) = +)$ or $Prob(M(x) = -)$ for short, respectively.

A probabilistic Turing machine M for a language L is said to have (two-sided) error probability ε if and only if for all $x \in \Sigma^*$,

$$\text{(a)} \qquad x \in L \implies Prob(M(x) = +) \geq 1 - \varepsilon,$$
$$\text{(b)} \qquad x \notin L \implies Prob(M(x) = +) \leq \varepsilon.$$

If a probabilistic Turing machine M for a language L satisfies that either for all $x \in \Sigma^*$,

$$\text{(a}') \qquad x \in A \implies Prob(M(x) = +) = 1$$

and (b) holds, or that for all $x \in \Sigma^*$, (a) and

$$\text{(b}') \qquad x \notin A \implies Prob(M(x) = +) = 0$$

holds, we say that M has one-sided error probability of ε. And clearly, if both, (a') and (b') hold, M is as good as a deterministic Turing machine. However, note that the notion of zero error is different from that. We will comment on this in more detail in Section 8.

Observe that due to our assumptions regarding the general behavior of a probabilistic Turing machine M on input x, i.e., a uniform distribution of the series of coin flips y and all series of coin flips having the same length depending solely on the length of the input, it follows that the probability $Prob(M(x) = +)$ is simply the number of series of coin flips leading to acceptance divided by the overall number of potential series of coin flips.

We are now prepared to define the basic acceptance criteria for probabilistic Turing machines:

Definition 1. 1. A language L belongs to the class NP (Nondeterministic Polynomial Time) if and only if there exists a probabilistic Turing machine M such that for all $x \in \Sigma^*$ the following holds:

$$x \in L \iff Prob(M(x) = +) > 0.$$

2. A language L belongs to the class PP (Probabilistic Polynomial Time) if and only if there exists a probabilistic Turing machine M such that for all $x \in \Sigma^*$ the following holds:

$$x \in L \iff Prob(M(x) = +) > \frac{1}{2}.$$

3. A language L belongs to the class BPP (Bounded-Probability Polynomial Time) if and only if there exists a probabilistic Turing machine M such that for all $x \in \Sigma^*$ the following holds:

$$x \in L \implies Prob(M(x) = +) \geq \frac{3}{4} \qquad \text{and}$$
$$x \notin L \implies Prob(M(x) = +) < \frac{1}{4}.$$

4. A language L belongs to the class RP (Random Polynomial Time) if and only if there exists a probabilistic Turing machine M such that for all $x \in \Sigma^*$ the following holds:

$$x \in L \implies Prob(M(x) = +) > \frac{1}{2} \qquad \text{and}$$
$$x \notin L \implies Prob(M(x) = +) = 0.$$

5. A language L belongs to the class ZPP (Zero-Error Probabilistic Polynomial Time) if and only if $L \in \text{RP} \cap \text{coRP}$.

The clases PPP and BPP are examples of probabilistic complexity classes with two-sided error. In terms of algorithms we may say that problems from BPP and PP admit polynomial-time bounded Monte Carlo algorithms. The complexity classes RP and coRP are probabilistic classes with one-sided error.

3 Basic Containments

In this section we will prove the known containments among the above defined probabilistic complexity classes. Observe that a result $\mathcal{C} \subseteq \mathcal{D}$ can be interpreted as any problem admitting a randomized algorithm of type $\mathcal{C}$ also admits a randomized algorithm of type $\mathcal{D}$.

Theorem 2. *1.* $\text{ZPP} \subseteq \text{RP} \subseteq \text{NP} \subseteq \text{PP}$.

2. $\text{RP} \subseteq \text{BPP} \subseteq \text{PP}$.

3. $\text{ZPP} \subseteq \text{coRP} \subseteq \text{BPP}$.

Proof. The inclusions $\text{ZPP} \subseteq \text{RP} \subseteq \text{NP}$ and $\text{ZPP} \subseteq \text{coRP}$ are immediate consequences of Definition 1. The inclusion $\text{BPP} \subseteq \text{PP}$ also follows immediately from Definition 1, since BPP has a two-sided error probability less then 0.25, whereas PP allows error probabilities arbitrarily close to 0.5.

For the inclusion $\text{NP} \subseteq \text{PP}$ let $L \in \text{NP}$ and M be a probabilistic Turing machine such that for all $x \in \Sigma^*$:

$$x \in L \iff Prob(M(x) = +) > 0.$$

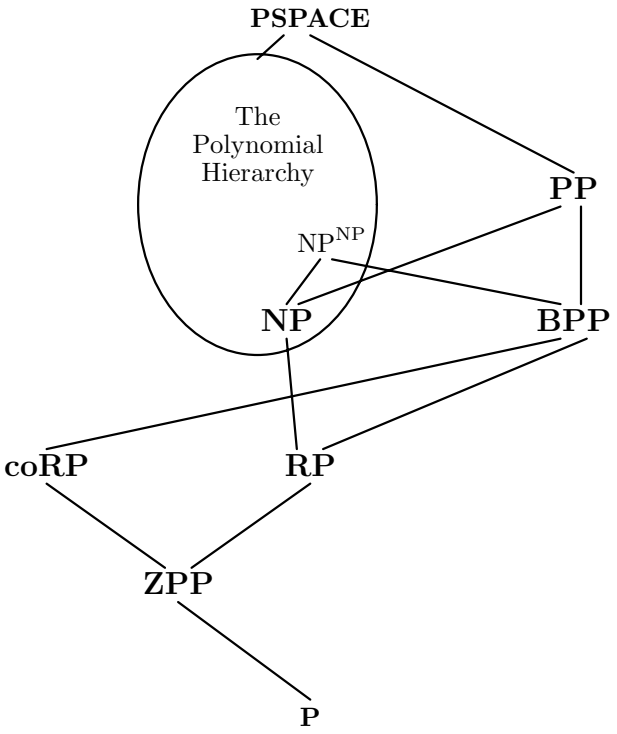

Figure 1: The inclusion structure of the probabilistic complexity classes. For comparison, the well-known Polynomial Hierarchy as well as the classes P, $\mathrm{NP}^{\mathrm{NP}}$ (the second level of the Polynomial Hierarchy, also known as Σ_2^{p}), and PSPACE have been added.

Consider the following probabilistic Turing machine M': On input $x \in \Sigma^*$, the Turing machine M‘ simulates the work of M on input x until $M(x)$ reaches a final state. If M accepts x then M' flips a random coin and accepts x regardless of the outcome of the coin flip. If M rejects x then M' flips a coin and rejects x if the coin flip returned 0 and accepts x if the coin flip returned 1.

Let p be a polynomial bounding the runtime of M. It follows that the polynomial q, where $q(n) = p(n) + 1$ for all n, bounds the runtime of M'. Furthermore, for all $x \in \Sigma^*$ we have that $x \in L$ if and only if the number of series of coin flips of $M(x)$ leading to acceptance is at least 1, and hence $x \in L$ if and only if the number of series of coin flips of $M'(x)$ leading to acceptance is at least $\frac{1}{2}(2^{p(|x|}-1)+2$. It follows that for all $x \in \Sigma^*$:

$$\begin{aligned} x \in L &\iff Prob(M(x) = +) > 0, \\ &\iff Prob(M'(x) = +) > \frac{1}{2}. \end{aligned}$$

This shows $L \in \mathrm{PP}$.

It remains to show $\mathrm{RP} \subseteq \mathrm{BPP}$ as well as $\mathrm{coRP} \subseteq \mathrm{BPP}$. We will only show $\mathrm{RP} \subseteq \mathrm{BPP}$ here, since $\mathrm{coRP} \subseteq \mathrm{BPP}$ then follows from the fact that BPP is closed under complement (see Section 6).

Suppose $L \in \mathrm{RP}$, hence there exists a probabilistic Turing machine M satisfying for all $x \in \Sigma^*$,

$$x \in L \implies Prob(M(x) = +) > \frac{1}{2},$$
$$x \notin L \implies Prob(M(x) = +) = 0.$$

Define a probabilistic Turing machine M' as follows: On input $x \in \Sigma^*$, simulate $M(x)$ until it reaches a final state and then simulate $M(x)$ again. Accept if and only if at least one of the two simulations of $M(x)$ ended in the final state "Accept." Otherwise reject.

Note that if $x \notin L$, then $M(x)$ does not reach "Accept" for all series of coin flips. Hence $M'(x)$ implicitly depending on two series of coin flips from the two simulations of $M(x)$ will not accept. If $x \in L$, we have by definition $Prob(M(x) = -) < \frac{1}{2}$ and hence, since $M'(x)$ simulates two independent runs of $M(x)$ and rejects if and only if both those simulations return "Reject," we obtain $Prob(M'(x) = -) < \frac{1}{2} \cdot \frac{1}{2}$. It follows that for all $x \in \Sigma^*$ we have

$$x \in L \implies Prob(M(x) = +) \geq \frac{3}{4},$$
$$x \notin L \implies Prob(M(x) = +) = 0.$$

This shows $L \in \mathrm{BPP}$. □

From the perspective of practicality, BPP is the most interesting class. BPP algorithms give correct answers with high probability, and as we will soon see the error probability can without loss of generality be quite close to 0. Of course RP algorithms are even better, since they offer a one-sided zero-error probability. However, for practical purposes, BPP algorithms are the best one can hope for. The bad news is that NP does not seem to be a subset of BPP and so it seems that not all NP problems admit efficient randomized algorithms. In fact, up to date no BPP-algorithm is known for any NP-complete problem and it appears that none will be found in the (near) future (see Section 6).

4 The Class NP – Nondeterministic Polynomial Time

Even though we made the connection to probabilistic classes very clear, the class NP is generally not considered to be a probabilistic complexity class. This is mainly due to the fact that the class NP comes with one-sided zero-error probability but the error probability (of the "other" side) can be arbitrarily close to

1. So, when viewing the class NP as a class of problems admitting randomized algorithms, it is clear that NP algorithms are extremely weak randomized algorithms of no practical use. Better descriptions of the class NP are the class of search problems or the class with short and easy to check proofs.

However, the class NP is well-studied and complexity theory in some sense evolved from attempts to solve the famous "P = NP?" question and to understand why that question has no easy answer and no easy proof for an answer either. Since many computational problems (from mathematics, biology, chemistry,...) belong to the group of hardest problems in NP, so-called NP-complete problems (see [GJ79]), the relation of the class NP to the probabilistic complexity classes has immediate consequences towards the existence of efficient randomized algorithms for those problems. Unfortunately, since it seems highly unlikely that NP $\subseteq$ BPP, the search for efficient randomized algorithms for NP-complete problems is somewhat hopeless in general.

5 The Class PP – Probabilistic Polynomial Time

The type of randomized algorithms that defines the class PP has a two-sided error probability arbitrarily close to 0.5. Even though PP algorithm are not very useful in practice, the study of the complexity class PP has led to many insights into the nature of probabilistic complexity classes and randomized algorithms. With respect to Figure 1 we mention the following inclusion.

Theorem 3. PP $\subseteq$ PSPACE.

The class PP is surprisingly powerful and it turns out that the type of randomization captured by the class PP together with so-called Turing reductions is strong enough to contain all of the Polynomial Hierarchy [Tod91]. The class PP has a number of interesting properties and connections to other probabilistic classes. As we have seen, PP contains all classes mentioned in this introduction, in particular NP and BPP. In the context of this paper, the acceptance criteria for PP is the weakest among the criteria mentioned in Definition 1 and thus emphasizes the fact that PP algorithms are useless in practice.

We mention in passing that PP possesses a number of strong closure properties such as closure under union, intersection, symmetric difference, and complement as well as closure under truth-table reductions. The class PP is also a prominent example of so-called counting classes and many computational problems based on counting solutions are many-one complete for PP. A prominent example of a many-one complete problem for the class PP is MAJORITY SAT, the set of all boolean formulas such that more than half of all its assignments are satisfying.

With respect to counting classes the acceptance criteria for PP is of the threshold type as is illustrated by the following theorem.

Theorem 4. *Let L be a language. The following statements are equivalent:*

1. $A \in \mathrm{PP}$.
2. *There exists a NPTM M and polynomial-time computable function f such that for all $x \in \Sigma^*$, $x \in A \iff M(x)$ has at least $f(x)$ accepting paths.*

6 The Class BPP – Bounded-Probability Polynomial Time

From a standpoint of practicality, BPP algorithms are true randomized algorithms. It is even conjectured that BPP = P, that is, any problem admitting a good randomized algorithm can be solved efficiently with a deterministic algorithm. We would like to mention two facts supporting that conjecture. The first fact is that the computational problem PRIMES, the problem of deciding if a given natural number is prime or not, that was long the prime example for problems in BPP, was shown to be in P by Agrawal, Kayal, and Saxena [AKS04]. The second fact, stated as Theorem 5, is that even though BPP problems admit by definition only algorithms with two-sided error probability (of 0.25) that error probability can be made quite small and hence, from a practical point of view, BPP problems admit algorithms that are nearly as good as deterministic algorithms.

Theorem 5. *Let $L \in \mathrm{BPP}$ and p be a polynomial. There exists a probabilistic Turing machine N such that for all $x \in \Sigma^*$ the following holds:*

$$x \in L \implies Prob(N(x) = +) \geq 1 - \frac{1}{2^{p(|x|)}},$$
$$x \notin L \implies Prob(N(x) = +) < \frac{1}{2^{p(|x|)}}.$$

Proof. The core idea of the proof is that by iterated applications of the original BPP algorithm for the language L we can make the error probability smaller and smaller. This technique is also known as the probability amplification technique.

Let $L \in \mathrm{BPP}$ and let p be a polynomial. Let M be an NPTM accepting L in the sense of BPP (see Definition 1).

Let $t \geq 1$ and consider the following NPTM M_t: On input $x \in \Sigma^*$, the NPTM M_t starts simulating the work of M on input x including flipping random coins. After finishing the simulation of M on input x, M_t starts another simulation of M on input x and does so t times. M_t accepts the input x if and only if out of the t simulations of M on input x at least $\frac{t}{2}$ were accepting computations.

Claim: For all $t \geq 1$ and all $x \in \Sigma^*$:

$$x \in L \implies Prob(M_t(x) = +) \geq 1 - \frac{1}{2}\left(\frac{3}{4}\right)^{\frac{t}{2}},$$
$$x \notin L \implies Prob(M_t(x) = +) < \frac{1}{2}\left(\frac{3}{4}\right)^{\frac{t}{2}}.$$

The claim follows from the following argument. Let $t \geq 1$. Define

$$p_{t,i} = \binom{t}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{t-i}$$

for all $0 \leq i \leq t$ and note that:

$$\begin{aligned} p_{t,i} &\leq \binom{t}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{t-i} \frac{\left(\frac{3}{4}\right)^{\frac{t}{2}-i}}{\left(\frac{1}{4}\right)^{\frac{t}{2}-i}} \\ &\leq \binom{t}{i} \left(\frac{3}{4}\right)^{\frac{t}{2}} \left(\frac{1}{4}\right)^{\frac{t}{2}} \\ &\leq \binom{t}{i} \left(\frac{3}{16}\right)^{\frac{t}{2}}. \end{aligned}$$

The probability p_t that a random path of $M_t(x)$ returns $c_L(x)$ is by definition of M_t the probability that out of the t simulations of $M(x)$ at least $\frac{t}{2}$ return $c_L(x)$. The latter can be bounded from below by

$$p_t \geq 1 - \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} p_{t,i}.$$

Using the inequalities from above we obtain:

$$\begin{aligned} p_t &\geq 1 - \sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} \binom{t}{i} \left(\frac{3}{16}\right)^{\frac{t}{2}} \\ &\geq 1 - \left(\frac{3}{16}\right)^{\frac{t}{2}} 2^{t-1} \\ &\geq 1 - \frac{1}{2} \left(\frac{3}{4}\right)^{\frac{t}{2}}. \end{aligned}$$

This completes the proof of the claim.

In order to prove the theorem we chose N to be a Turing machine that on input x is essentially M_t on input x for a carefully chosen t. In particular we want to have

$$1 - \frac{1}{2} \left(\frac{3}{4}\right)^{\frac{t}{2}} \geq 1 - \frac{1}{2^{p(|x|)}}$$

and thus need $t \geq 2\frac{1-p(|x|)}{\log\left(\frac{3}{4}\right)}$. Note that t depends on $|x|$, yet a suitable t satisfying that inequality can be computed in polynomial time. □

We mention that the class BPP is closed under union, intersection and complement. On the other hand, even though BPP is closed under many-one reductions, it might not have many-one complete problems, a property that BPP shares with the probabilistic complexity classes RP and ZPP.

Finally, we turn to a rather bad news. The study of probabilistic complexity classes led to the insight that many NP problems might not have good randomized algorithms. This follows from the fact that all problems from BPP have polynomial-size circuit families which in turn can be used to show that an containment NP $\subseteq$ BPP is highly unlikely, it would imply a collapse of the polynomial hierarchy to its second level. Another interesting consequence of an assumption NP $\subseteq$ BPP is, in the context of this paper, stated in the next theorem.

Theorem 6. NP $\subseteq$ BPP $\implies$ NP $\subseteq$ RP.

7 The Class RP – Random Polynomial Time

Even better than BPP algorithms, RP algorithms not only have, as we will soon see, very small error probability, but they like NP algorithms will never falsely accept strings.

Note that the 0.5 success probability in case of acceptance is chosen at will, any other success probability that is polynomially related to the length of the input would lead to the same complexity class.

Theorem 7. *A language L is in RP if and only if there exists a probabilistic Turing machine M and a polynomial q such that for all $x \in \Sigma^*$,*

$$x \in L \implies Prob(M(x) = +) > \frac{1}{q(|x|)},$$
$$x \notin L \implies Prob(M(x) = +) = 0.$$

Proof. Clearly, if $L \in$ RP then there exists a probabilistic Turing machine M accepting L in the sense of Definition 1. Obviously M together with the polynomial q, where $q(n) = 2$ for all n, satisfies for all $x \in \Sigma^*$,

$$x \in L \implies Prob(M(x) = +) > \frac{1}{q(|x|)},$$
$$x \notin L \implies Prob(M(x) = +) = 0.$$

Now let $L \subseteq \Sigma^*$ and let M be a probabilistic Turing machine and q be a polynomial such that for all $x \in \Sigma^*$,

$$x \in L \implies Prob(M(x) = +) > \frac{1}{q(|x|)},$$
$$x \notin L \implies Prob(M(x) = +) = 0.$$

Similarly to the proof of the claim inside the proof of Theorem 5 we can show $L \in$ RP by repeated simulations of M. In contrast to that earlier proof we

accept a string x if and only if at least one of the simulations of $M(x)$ led to acceptance. More formally, for all $t \geq 1$ define the probabilistic Turing machine M'_t as follows: On input $x \in \Sigma^*$, the Turing machine M'_t simulates the work of M on input x including flipping random coins. After finishing the simulation of M on input x, M'_t starts another simulation of M on input x and so on until $M(x)$ has been simulated t times. M'_t accepts the input x if and only if out of the t simulations of M on input x at least one was an accepting computation.

Observe that in case $x \notin L$, M'_t will not accept x regardless of the random choices made during the simulations of $M(x)$. For the case $x \in L$, observe that the probability that $M(x)$ rejects x is smaller than or equal to $1 - \frac{1}{q(|x|)}$. Hence, the probability that M'_t rejects x even though $x \in L$ is at most $\left(1 - \frac{1}{q(|x|)}\right)^t$. But note that as t grows, that term goes to 0, and so for t large enough we have $\left(1 - \frac{1}{q(|x|)}\right)^t < \frac{1}{2}$. It remains to show that t is polynomial in $|x|$. However, this essentially follows from the rapid convergence $\lim_{k\to\infty}\left(1 - \frac{1}{k}\right)^k = \frac{1}{e}$ where $e = 2,7182818284\ldots$. □

Finally we mention that RP is closed under union and intersection, yet seems to lack many-one complete languages.

8 The Class ZPP – Zero-Error Probabilistic Polynomial Time

The complexity class ZPP is defined as the intersection of RP and its complementary class coRP. Since both RP and coRP come with one-sided zero-error probability, one can show that ZPP has two-sided zero-error probability. However, note that two-sided zero-error probability algorithms for a language L simply guarantee that they on input x:

1. return "Accept" with high probability if $x \in L$,
2. return "Reject" with high probability if $x \notin L$,
3. never return "Accept" if indeed $x \notin L$, and
4. never return "Reject" if indeed $x \in L$.

Still, two-sided zero-error probability algorithms are not identical to deterministic algorithms, since they might return neither "Accept" nor "Reject," but rather "Undecided." Randomized algorithms that never return a wrong answer, yet return the correct answer with high probability are known as Las Vegas algorithms.

The following theorem states that fact in more detail.

Theorem 8. *Let $L \subseteq \Sigma^*$. The language L belongs to ZPP if and only if there exists a probabilistic polynomial-time Turing machine M having three types of final states, "Accept" (+), "Reject" (−) and "Undecided" (u), such that the following holds for all $x \in \Sigma^*$:*

$$x \in L \implies Prob(M(x) = +) > \frac{1}{2} \ \wedge \ Prob(M(x) = -) = 0,$$
$$x \notin L \implies Prob(M(x) = +) = 0 \ \wedge \ Prob(M(x) = -) > \frac{1}{2}.$$

Proof. Let $L \subseteq \Sigma^*$ and suppose $L \in$ ZPP. It follows from Definition 1 that $L \in \mathrm{RP} \cap \mathrm{coRP}$ and hence there exist two probabilistic polynomial-time Turing machines N and N' that accept L and $\overline{L}$ in the sense of RP. Hence, for all $x \in \Sigma^*$ we have:

$x \in L \implies Prob(N(x) = +) > \frac{1}{2}$,
$x \notin L \implies Prob(N(x) = +) = 0$ (or equivalently $Prob(N(x) = -) = 1$),
$x \in \overline{L} \implies Prob(N'(x) = +) > \frac{1}{2}$, and
$x \notin \overline{L} \implies Prob(N'(x) = +) = 0$ (or equivalently $Prob(N'(x) = -) = 1$).

We define a probabilistic Turing machine M as follows:

1. On input $x \in \Sigma^*$, the Turing machine M simulates the work of $N(x)$ until it reaches a final state "Accept" (+) or "Reject" (−) and afterwards simulates the work of $N'(x)$ until that machine reaches a final state "Accept" (+) or "Reject" (−).

2. If the final states reached by $N(x)$ and $N'(x)$ are + and −, respectively, $M(x)$ accepts. If the final states reached by $N(x)$ and $N'(x)$ are − and +, respectively, $M(x)$ rejects. If the final states reached by $N(x)$ and $N'(x)$ are − and −, respectively, $M(x)$ finishes its computation in the final state "Undecided."

It is not hard to verify that for all $x \in \Sigma^*$ we have:

$$x \in L \implies Prob(M(x) = +) > \frac{1}{2} \ \wedge \ Prob(M(x) = -) = 0,$$
$$x \notin L \implies Prob(M(x) = +) = 0 \ \wedge \ Prob(M(x) = -) > \frac{1}{2}.$$

Now suppose that for a language $L \in \Sigma^*$, we have a probabilistic polynomial-time Turing machine M having three types of final states, "Accept" (+), "Reject" (−) and "Undecided" (u), such that the following holds for all $x \in \Sigma^*$:

$$x \in L \implies Prob(M(x) = +) > \frac{1}{2} \ \wedge \ Prob(M(x) = -) = 0,$$
$$x \notin L \implies Prob(M(x) = +) = 0 \ \wedge \ Prob(M(x) = -) > \frac{1}{2}.$$

We define two Turing machines N and N' as follows: The Turing machine N on input $x \in \Sigma^*$ simulates the work of $M(x)$ and accepts if $M(x)$ accepts, and rejects if $M(x)$ reaches the final state "Undecided." The Turing machine N' on input $x \in \Sigma^*$ simulates the work of $M(x)$ and accepts if $M(x)$ rejects, and rejects if $M(x)$ reaches the final state "Undecided."

It is not hard to verify that N and N', respectively, accepts the languages L and $\overline{L}$ in the sense of RP, and hence $L \in \mathrm{RP} \cap \mathrm{coRP}$. □

Regarding closure properties, we note that ZPP is closed under union, intersection, and complement.

9 Historical Remarks and Literature Pointers

It is interesting to note that the study of probabilistic Turing machines is closely connected to the search for fast algorithms for primality testing. The famous Miller–Rabin Test (see [Rab76, Mil76]) for primality in some sense marks the beginning of the complexity-theoretic study of randomized algorithms. The class RP was introduced by Adleman and Manders [AM77]. The famous Solovay–Strassen Test [SS77] for primality essentially shows PRIMES $\in$ coRP. Later, Adleman and Huang [AH87] being influenced by the work of Goldwasser and Kilian [GK86] showed PRIMES $\in$ RP which together with the Solovay-Strassen result implies PRIMES $\in$ ZPP. The result of Agrawal, Kayal, and Saxena [AKS04] that showed PRIMES $\in$ P concludes that line of research.

An early model of probabilistic Turing machine can be found in [dLMSS56]. The notion of probabilistic Turing machines as well as the probabilistic complexity classes were formally introduced by Gill [Gil77]. The characterization of ZPP contained in Theorem 8 can also be found in [Gil77]. The containment of BPP in the Polynomial Hierarchy was shown by Sipser [Sip83] and was improved by Gács (see [Sip83]) to BPP $\subseteq \Sigma_2^{\mathrm{p}} \cap \Pi_2^{\mathrm{p}}$ (see also [Lau83] for a simple proof of the latter result).

Characterizations of PP as threshold or counting classes were obtained and studied in [Wag86]. Toda [Tod91] showed that the Polynomial Hierarchy is contained in P^{PP}, the set of all problems being Turing-reducible to PP. The result is also known as Toda's Theorem.

Hints that ZPP, RP, and BPP do not have many-one complete languages are contained in [Adl78, Sip82]. Adleman [Adl78] showed that all problems from RP have polynomial-size circuits. A similar result for BPP is contained in [BG81]. Karp and Lipton [KL80] showed that NP having polynomial circuits implies a collapse of the Polynomial Hierarchy to it's second level and thus making an inclusion NP $\subseteq$ BPP highly unlikely. Theorem 6 was proven by Ko [Ko82]. Some relativization results regarding probabilistic complexity classes can be found in [Rac82].

References

[Adl78] L. Adleman. Two theorems on random polynomial time. In *Proceedings of the 19th IEEE Symposium on Foundations of Computer Science*, pages 75–83, 1978.

[AH87] L. Adleman and M. Huang. Recognizing primes in random polynomial time. In *Proceedings of the 19th ACM Symposium on Theory of Computing*, pages 462–469. ACM Press, May 1987.

[AKS04] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[AM77] L. Adleman and K. Manders. Reducibility, randomness, and intractibility. In *Proceedings of the 9th ACM Symposium on Theory of Computing*, pages 151–153. ACM Press, 1977.

[BG81] C. Bennett and J. Gill. Relative to a random oracle A, $\mathrm{P}^A \neq \mathrm{NP}^A \neq \mathrm{coNP}^A$ with probability 1. *SIAM Journal on Computing*, 10(1):96–113, 1981.

[dLMSS56] K. de Leeuw, E. Moore, C. Shannon, and N. Shapiro. *Computability by Probabilistic Machines*. Princeton University Press, Princeton, New Jersey, 1956.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[GK86] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proceedings of the 18th ACM Symposium on Theory of Computing*, pages 316–329. ACM Press, 1986.

[KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th ACM Symposium on Theory of Computing*, pages 302–309. ACM Press, April 1980. An extended version has also appeared as: Turing machines that take advice, *L'Enseignement Mathématique*, 2nd series 28, 1982, pages 191–209.

[Ko82] K. Ko. Some observations on the probabilistic algorithms and NP-hard problems. *Information Processing Letters*, 14(1):39–43, 1982.

[Lau83] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.

[Mil76] G. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, 13:300–317, 1976.

[Pap94] C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Rab76] M. Rabin. Probabilistic algorithms. In *Algorithms and Complexity*. Academic Press, 1976.

[Rac82] C. Rackoff. Relativized questions involving probabilistic algorithms. *Journal of the ACM*, 29(1):261–268, 1982.

[Sip82] M. Sipser. On relativization and the existence of complete sets. In *Proceedings of the 9th International Colloquium on Automata, Languages, and Programming*. Springer-Verlag *Lecture Notes in Computer Science #140*, 1982.

[Sip83] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 330–335. ACM Press, 1983.

[SS77] R. Solovay and V. Strassen. A fast Monte Carlo test for primality. *SIAM Journal on Computing*, 6:84–85, 1977. Erratum appears in the same journal, 7(1):118, 1978.

[Tod91] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[Wag86] K. Wagner. Some observations on the connection between counting and recursion. *Theoretical Computer Science*, 47:131–147, 1986.