Randomized Complexity

QIN Teng

December 17, 2010

Abstract

In this report, the concept of randomized algorithms is firstly motivated. Following these examples, we formulated and discussed about the definition and properties of randomized Turing Machines and complexity classes. Finally, we proposed some results about the relationship between randomized complexity classes and other complexity classes.

Contents

1 Motivations **2**
- 1.1 Polynomial Identity Testing 2
- 1.2 Probabilistic Primality Testing 3

2 Randomized Complexity **6**
- 2.1 Randomized Turing Machine 6
- 2.2 The Class **BPP** 6
- 2.3 One-sided Error 8

3 Complexity Classification **9**
- 3.1 Among Randomized Complexity Classes 9
- 3.2 $\mathbf{BPP} \subset \Sigma_2^p \cap \Pi_2^p$ 10

References **12**

1 Motivations

Most programming languages provide a built-in (pseudo-)random number generator for making random choices during the computation. Although the existence of true randomness during computation is still being debated by scientists and philosophers, these random tools does helped us a lot. In the following of this paragraph we can see two examples about how randomness can be used in algorithms.

1.1 Polynomial Identity Testing

First, we are going to consider a simple problem: given two polynomials over $\mathbb{Z}$, determine whether they are the same polynomial. Or we can describe the problem as, given a polynomial, determine whether it is identical to zero. Note that the polynomials might be given in implicit form, that mean, given by result of complicated calculations between polynomials. That makes the problem much harder since we can not directly compare between the coefficients and zero, and actually there has no known efficient deterministic algorithm for that.[4] Nevertheless, when value of every variable in the polynomial is given, it is not hard to evaluate it even in implicit form. Therefore we can attack the problem using a very simple algorithm, that we randomly choose several values of variables in the polynomial, evaluate the polynomial and see the result is zero or not. However, if our randomly chosen values coincidentally being the root of the polynomial, we could wrongly recognized a non-zero polynomial as zero ones. We could pick more values to reduce the odds of such mistakes, but since the number of roots of a multi-variable polynomial might be infinite, there is no way to eliminate such situations. Luckily, we have the following lemma:

Lemma 1 (Schwartz-Zippel's Lemma)**.** If a polynomial $p(x_1, \cdots, x_m)$ over $F = GF(q)$ is nonzero and has total degree at most d, then

$$\texttt{Pr}[p(a_1, \cdots, a_m) \neq 0] \geq 1 - \frac{d}{q}$$

where the probability is over all choice of $a_1, \cdots, a_m \in F$.

Proof. We proof the lemma by induction over m.If $m = 1$, the lemma is true according to the Fundamental Theorem of Algebra. Suppose it is true for the number of variables less or equals to $m - 1$. Then we can write p as

$$p(x_1, \cdots, x_m) = \sum_{i=0}^{d} x_1^i p_i(x_2, \cdots, x_m)$$

in which we can see that the degree of p_i is at most $d - i$. Since p is not identical to zero, at least one p_i is nonzero. Therefore we pick i be the largest index that p_i is nonzero. Then by the inductive hypothesis we have

$$\texttt{Pr}_{a_2,\cdots,a_m}[p_i(a_2, \cdots, a_m) \neq 0] \geq 1 - \frac{d-i}{q}$$

for all $p_i \neq 0$. Since $p(x_1, a_2, \cdots, a_m)$ is nonzero univariate polynomial of degree i, we know that it can become 0 for at most i values of x_1. Hence

$$\texttt{Pr}[p(a_1, \cdots, a_m) \neq 0] \geq (1 - \frac{i}{q})(1 - \frac{d-i}{q}) \geq 1 - \frac{d}{q}$$

$\square$

From the lemma we know that, the probability of our algorithm making mistakes is bounded. Hence we have a polynomial time algorithm that, when it says no, we know that the original polynomial is definitely non-zero, when it says yes, there is a small chance that the original polynomial is actually non-zero, and we can make the probability of error being arbitrarily small while remaining polynomial running time. Thus we can say although the algorithm is not perfect, but it is a good enough approach for the problem compared with naive exponential algorithm, and such improvement is just came from the usage of random numbers.

1.2 Probabilistic Primality Testing

Now we are going to show another randomized algorithm that is not such trivial. The problem is also quite fundamental, that is, given a number n(in the form of binary string), determine whether it is prime. The naive Eratosthenes' Algorithm checks all possible primes less than $\sqrt{n}$, thus takes exponential time since the length of binary string of n is only $\log n$. Recently there are polynomial algorithm proposed[6], but we are still going to introduce an efficient randomized algorithm that is also of polynomial time.

The algorithm mainly based on following theorem.

Theorem 2 (Fermat's Little Theorem)**.** If p is a prime number, and $a \in \mathbb{Z}_p^+$, then we have

$$a^{p-1} \equiv 1 (\text{mod } p)$$

This theorem provided a method to "test" the prime number, we call it the Fermat's Test. Denote that p passes the Fermat's Test under a means $a^{p-1} \equiv 1(\text{mod } p)$, and we say a number is a pseudo-prime number if it passes

Fermat's Test under all number that is smaller than and relatively prime to it. From Theorem 2 we know that all prime number is also pseudo-prime. However, the reverse is not true for some pseudo-prime number that is actually not prime, called Carmichael's Numbers. Also it still takes exponential time to check all possible a to determine if p is pseudo-prime. Therefore we develop the following lemma.

Lemma 3. For any interger $p > 1$, if p isn't pseudo-prime, then p failed at least half of Fermat's Test based on numbers in $\mathbb{Z}_p^+$.

Proof. We call a a witness for p if $a^{p-1} \not\equiv 1 (\text{mod } p)$. Let $\mathbb{Z}_p^*$ be all integers less than and relatively prime to p. If p is not a pseudo-prime, it has a witness $a \in \mathbb{Z}_p^*$.

If $d \in \mathbb{Z}_p^*$ is not a witness, we have $d^{p-1} \equiv 1(\text{mod } p)$, and hence $(da \text{ mod } p) \not\equiv 1(\text{mod } p)$, which means $(da \text{ mod } p)$ is a witness. For non-witness $d_1, d_2 \in \mathbb{Z}_p^*$, we have $(d_1 a \text{ mod } p) \neq (d_2 a \text{ mod } p)$ since otherwise $(d_1 - d_2)a \equiv 0(\text{mod } p) \Rightarrow \exists c \in \mathbb{Z}, (d_1 - d_2)a = cp$. But $(d_1 - d_2) < p$, so $a = \frac{cp}{d_2 - d_2}$ and p have a common factor greater than 1, which is a contradiction since ever number in $\mathbb{Z}_p^*$ is relatively prime to p. This shows at lease half of numbers in $\mathbb{Z}_p^*$ is witness for p.

Finally we show that, every number b that is in $\mathbb{Z}_p^+$ but not in $\mathbb{Z}_p^*$, which means not relatively prime to p, is a witness. In fact, for every $e > 0$, b^e is also not relatively prime to p, which means $b^{p-1} \not\equiv 1(\text{mod } p)$. That proved our original lemma. $\square$

This lemma shows that the probability of a number p that is not pseudo prime passes the Fermat's Test under arbitrarily picked a from $\mathbb{Z}_p^+$ is less than $\frac{1}{2}$, which means if we pick k numbers from $\mathbb{Z}_p^+$, the probability would be less than $\frac{1}{2^k}$. Since a^{p-1} can be calculated in polynomial time[7], we got a good enough polynomial time randomized algorithm to determine pseudo-primality.

To eliminate the effect of Carmichael's Numbers, we note the following truth.

Lemma 4. If $\exists b \in \mathbb{Z}_p^+, b \not\equiv \pm 1(\text{mod } p), b^2 \equiv 1(\text{mod } p)$, then p is composite.

Proof. From the condition we have $b^2 - 1 \equiv 0(\text{mod } p)$, therefore

$$(b-1)(b+1) \equiv 0(\text{mod } p)$$

hence

$$\exists c \in \mathbb{Z}, (b-1)(b+1) = cp$$

Since $b \not\equiv 1 (\mathrm{mod}\ p)$, so $0 < b-1 < p, 0 < b+1 < p$. But the multiple of a prime number can not be written as product of number that is less than it, therefore p is composite. □

For each a that p passes the Fermat's Test under it, we have $a^{p-1} \equiv 1 (\mathrm{mod}\ p)$. Therefore if we write $p-1 = st$ where s is a odd number and $t = 2^h$, we have $a^{s\cdot 2^0} \equiv 1 (\mathrm{mod}\ p), \cdots, a^{s\cdot 2^h} \equiv 1 (\mathrm{mod}\ p)$. So if any of these h numbers have $a^{s\cdot 2^i} \not\equiv \pm 1 (\mathrm{mod}\ p)$, from Lemma 4 we know that p is not prime.

The following lemma proved the effectiveness of this test.

Lemma 5. If p is a odd composite number, $a \in \mathbb{Z}_p^+$ is an arbitrarily picked number, then

$$\mathtt{Pr}[a \text{ is a witeness}] \geq \frac{1}{2}$$

Proof. To proof that, we only need to designate a witness to each non-witness among $\mathbb{Z}_p^+$. For each non-witness, we have the number sequence in previously described test is all 1s or contains -1. Pick h be a non-witness and let j be the position of -1 in the sequence, then $h^{s\cdot 2^j} \equiv -1 (\mathrm{mod}\ p)$. Since p is composite, it is either a power of some prime or $p = qr$ from some relatively prime q and r. The Chinese Reminder Theorem implies that $\exists t \in \mathbb{Z}_p^+$ such that

$$t \equiv h (\mathrm{mod}\ p)$$

and

$$t \equiv 1 (\mathrm{mod}\ r)$$

Therefore

$$t^{s\cdot 2^j} \equiv -1 (\mathrm{mod}\ q)$$

and

$$t^{s\cdot 2^j} \equiv 1 (\mathrm{mod}\ r)$$

So t induces a witness of p since $t^{s\cdot 2^j} \not\equiv \pm 1 (\mathrm{mod}\ p)$ but $t^{s\cdot 2^{j+1}} \equiv 1 (\mathrm{mod}\ p)$.

Then we prove that $dt \bmod p$ is unique witness for each non-witness d. In fact, for $d_1 \neq d_2$, since $t \cdot t^{s\cdot 2^{j+1}-1} \bmod p = 1$, therefore if $td_1 \bmod p = td_2 \bmod p$, we have

$$d_1 = t \cdot t^{s\cdot 2^{j+1}-1} d_1 \bmod p = t \cdot t^{s\cdot 2^{j+1}-1} d_2 \bmod p = d_2$$

On the other hand, if $p = q^2$ where q is a prime and $e > 1$, let $t = 1 + q^{e-1}$. By the Binomial Theorem, we have

$$t^p = (1 + q^{e-1})^p = 1 + pq^{e-1} + \text{multiples of higher power of } q^{e-1}$$

and equivalent to 1 mod p, which means t should fail p in the Fermat's Test since $t^p \equiv t \not\equiv 1 (\text{mod } p)$ if $t^{p-1} \equiv 1$. Then just like previous proof, if d is a non-witness, we have $d^{p-1} \equiv 1 (\text{mod } p)$ then dt mod p is a unique witness for it. Thus proved that there is more witnesses than non-witnesses in $\mathbb{Z}_p^+$ if p is composite, and shows our original lemma. □

Finally we have the polynomial time algorithm that, when it says p is not a prime, p is definitely not, otherwise if we tested p over k numbers, the probability of the algorithm making mistake is less than $\frac{1}{2^k}$, which means can be arbitrarily small while remaining polynomial time.

2 Randomized Complexity

2.1 Randomized Turing Machine

From the previous two examples, we can see the power of randomness in designing algorithms. However, the model that we used to describe computational complexity, the Turing Machine, is not equipped to describe such algorithms[1]. Therefore, we are going to define a variant of Turing machine that able to flip coins or roll dices.

Definition 6. A Probabilistic Turing Machine(PTM) is a Turing Machine that has two transition functions σ_0 and σ_1. For a PTM M and a input x, we choose each step with probability $\frac{1}{2}$ to σ_0 or σ_1. The computation ends when it reaches an accept status or a reject status. Denote $M(x) \in \{0, 1\}$ be the random variable generated by PTM M over input x.

2.2 The Class BPP

Having the probabilistic computing model, now we can define complexity classes over PTM. Like the class **P** in traditional Turing Machines, we aimed on capturing the efficient probabilistic computations and define the class **BPP**[2].

Definition 7 (**BPP**)**.** For $T : \mathbb{N} \to \mathbb{N}$ and $L \subset \{0, 1\}^*$ we say that a PTM M decides L in time $T(n)$ if for every $x \in \{0, 1\}^*$, M halts in $T(|x|)$ steps regardless of its random choices, and $\texttt{Pr}[M(x) \neq L(x)] < \frac{1}{3}$.

[1]Some may argue that in actual practice, all random number generator used in algorithm is still deterministic procedures. But note that we haven't used any specific description of a random number generator, but only abstract properties of probability, so the theoretical model used to analysis such algorithms should also be able to describe such ideal randomness.

[2]That means **B**ounded-error **P**robabilistic **P**olynomial time

We let **BPTIME**$(T(n))$ be the class of languages decided by PTMs in $O(T(n))$ time and define **BPP** $= \cup_c$**BPTIME**(n^c).

The strange part in the definition is the magic number $\frac{1}{3}$. It seems not good enough, and comes from nowhere. Now we are going to show by following lemma that this number is actually irrelevant. In fact, we can imagine that to run a PTM with higher error probability polynomial many times, and take the "voted result" among them to reduce the chance of error.

Lemma 8 (Amplification Lemma)**.** Let $0 < \epsilon < \frac{1}{2}$ be a fixed constant. Then for any polynomial $\texttt{Poly}(n)$, a probablistic polynomial time Turing Machine M_1 that operates with error probability ϵ has an equivalent probabilistic polynomial time Turing Machine M_2 that operates with an error probability of $2^{-\texttt{Poly}(n)}$.

Proof. We construct M_2 that calculates x as following:

1. Calculated k as following.
2. Run $2k$ independent simulations of M_1 on input x.
3. If most runs accepted, then accept, otherwise reject.

The correctness of M_2 depends on whether most of M_1 runs returns the correct answer. Now we try to bound the probability that at least half of those runs were wrong.

Let S be any sequence of results that M_2 obtained, and let p_S be the probability that M_2 obtains S. Suppose S has c correct results and w wrong result, we call S a bad sequence if $c \leq w$. Therefore $p_S \leq \epsilon^w(1-\epsilon)^c$, which is at most $\epsilon^k(1-\epsilon)^k$ since $k \leq w$ and $\epsilon < 1 - \epsilon$.

The sum of all p_S is the probability that M_2 goes wrong. We have 2^{2k} possible sequences, therefore we have at most such many bad sequences. Hence

$$\texttt{Pr}[M_2 \text{ outputs incorrectly}] = \sum_{\text{bad } S} p_S \leq 2^{2k} \cdot \epsilon^k(1-\epsilon)^k = (4\epsilon(1-\epsilon))^k$$

Since $\epsilon < \frac{1}{2}$, $4\epsilon(1-\epsilon) < 1$ and therefore the probability above decreases exponentially in k. So, for any given $t \geq 1$, we can bound the probability that M_2 goes wrong by letting $k \geq \frac{t}{\alpha}$ where $\alpha = \log_2(4\epsilon(1-\epsilon)$. This shows we obtained the error probability of $2^{-\texttt{Poly}(n)}$ within polynomial time. □

As we did in the deterministic situations, we can then define the reversed side complexity class **co-BPP** that contains languages whose complement is in **BPP**. It is also natural to define **BPP**-Completeness, but unfortunately we haven't found such problems till now[2].

Similarly to what we have did on defining the class **NP**, we can give an alternative definition for the class **BPP** using the concept of "evidences" or "proofs" on deterministic computation model, which indicates the sequence of coin toss.

Definition 9 (**BPP**-Alternative)**.** A language L is in **BPP** if there exists a polynomial-time Turing Machine M and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for every $x \in \{0,1\}*$

$$\mathtt{Pr}_{r \in_R \{0,1\}^{p(|x|)}}[M(x,r) \neq L(x)] < \frac{1}{3}$$

2.3 One-sided Error

Remind our previously described two probabilistic algorithms, note that they have a property in common is that when the algorithm says no about some decision problem, the answer to the problem is definitely being "no'". However, when the algorithm gives a positive answer, it might be wrong. We define following complexity class to capture such idea.

Definition 10. **RTIME**$(T(n))$ contains every language L for which there is a PTM M running in $T(n)$ time such that

$$x \in L \Rightarrow \mathtt{Pr}[M(x) = 1] \geq \frac{2}{3}$$

and

$$x \notin L \Rightarrow \mathtt{Pr}[M(x) = 1] = 0$$

We define $\mathbf{RP} = \cup_{c>0}\mathbf{RTIME}(n^c)$[3]

It is easy to see that the the language calculated by the probabilistic primality testing is in the class **RP**, which the other on, `ZERO-Poly`, is in the complement of it, the **co-RP**. Such algorithms that computes languages in **RP** is often called Las Vegas Algorithms, which means you have to gamble over on side. However, the algorithm corresponding to the class **BPP** is also commonly name after a name of gambling resort, the Monte Carlo Algorithms.

We could hence require that the randomized algorithm makes mistake in probability zero.

[3]Where **RP** means **R**andomized **P**olynomial time.

Definition 11. The class $\mathbf{ZTIME}(T(b))$ contains all the languages L for which there is a machine M that runs in an expected-time $O(T(n))$ such chat for every input x, when ever M halts on x, the output $M(x) = L(x)$.

We define $\mathbf{ZPP} = \cup_{c>0}\mathbf{ZTIME}(n^c)$[4]

3 Complexity Classification

3.1 Among Randomized Complexity Classes

[3] Now we are going to discuss about the connections between these randomized complexity classes. As we can see, the class **BPP** is aimed on giving a "randomized duplicate" of the traditional class **P**. Therefore we can easily have **BPP** = **co-BPP** from Definition 7, while the relationship between class **RP** and **co-RP** is still remain unknown just like the case of **NP** and **co-NP**. However, we have following theorem that is somehow surprising if compared with deterministic cases.

Theorem 12. **ZPP** = **RP** ∩ **co-RP**.

Proof. If we have a language L recognized by both a **RP** algorithm A and a **co-RP** algorithm B, then we proposed an algorithm runs over any input x as following

1. Run A over x. If A return yes, then the answer must be yes and halts.
2. Otherwise run B over x. If it returns no, then the answer must be no and halts.
3. If neither happens, repeat from Step 1.

From the property of **RP** and **co-RP** we know that A and B can not go both wrong, and the chance of the algorithm returns the wrong answer is less than $\frac{1}{2}$. That means the chance of running k round is exponentially decreasing to zero by k. Therefore we have our polynomial time Las Vegas Algorithm and **RP** ∩ **co-RP** ⊂ **ZPP**.

To show the inverse side inclusion, recall the Markov's Inequality

$$\mathtt{Pr}(|X| > a) \leq \frac{\mathtt{E}(|X|)}{a}$$

which means if we run our **ZPP** algorithm double of it's expecting running time, the chance we stop it before it actually halts is less than $\frac{1}{2}$. Then we

[4]**ZPP** means **Z**ero-error **P**robabilistic **P**olynomial time.

just answer no for such situation, and from Definition 10 we know it is a valid **RP** algorithm. The **co-RP** side can be proved similarly, which come our final conclusion. □

In the last, we have the obvious conclusion that

$$\mathbf{RP} \subset \mathbf{BPP}, \mathbf{co\text{-}RP} \subset \mathbf{BPP}$$

3.2 BPP $\subset \Sigma_2^p \cap \Pi_2^p$

We are now interested in the relationship between randomized complexity classes and deterministic complexity classes. Note that classical Turing Machine is actually PTM that flips no coins, thus

$$\mathbf{P} = \mathbf{co\text{-}P} = \mathbf{BPP} = \mathbf{co\text{-}BPP}$$

In fact, by the recent development in the research area of derandomization, many people believes that these classes are actually the same. It is also clear that

$$\mathbf{BPP} \subset \mathbf{EXP}$$

since we can enumerate all exponentially many coin flipping sequences. Also for the **RP** classes, we have

$$\mathbf{P} = \mathbf{co\text{-}P} \subset \mathbf{RP}, \mathbf{P} = \mathbf{co\text{-}P} \subset \mathbf{co\text{-}RP}$$

Also we can see that we could "guess" the coin flipping[5] sequence using non-deterministic Turing Machine, which means

$$\mathbf{RP} \subset \mathbf{NP}, \mathbf{co\text{-}RP} \subset \mathbf{co\text{-}NP}$$

Meanwhile the relationship between **BPP** and **NP** is still remaining unknown. However, we have the result for larger complexity classes, where we know that **NP** is Σ_1.

Theorem 13. BPP $\subset \Sigma_2^p \cap \Pi_2^p$

Proof. Since **BPP** is closed under complement, we only have to show that **BPP** $\subset \Sigma_2^p$.

Suppose a language $L \in$ **BPP**. By the alternative Definition 9 and Lemma 8, there exists a polynomial-time deterministic Turing Machine M for L that on input of length n uses $m = \texttt{Poly}(n)$ random bits and satisfies

$$x \in L \Rightarrow \texttt{Pr}_r[M(x,r) \text{ accepts}] \geq 1 - 2^n$$

[5]Or can be said as existence of "proof" or "evidence" for the input if using alternative definition for **NP**

and

$$x \notin L \Rightarrow \texttt{Pr}_r[M(x,r) \text{ accepts}] \leq 2^{-n}$$

For $x \in \{0,1\}^n$, let S_x be the set of r that M accepts the input pair $\langle x, r\rangle$. Then either $|S_x| \geq (1-2^{-n})2^m$, or $|S_x| \leq 2^{-n}2^m$ for $x \in L$ and $x \notin L$ respectively. Now we are going to check using two quantifiers which is true for two cases.

For a set $S \subset \{0,1\}^m$ and string $u \in \{0,1\}^m$, define $S+u = \{x+u : x \in S\}$ where $+$ means bitwise XOR. Let $k = \lceil \frac{m}{n} \rceil + 1$, we proof following two lemmas.

Lemma 14. $\forall S \subset \{0,1\}^m, |S| \leq 2^{m-n}$ and any k vectors $u_1, \cdots, u_k$

$$\cup_{i=1}^k (S+u_i) \neq \{0,1\}^m$$

Proof. Since $|S+u_i| = |S|$, we have, for sufficiently large n

$$| \cup_{i=1}^k (S+u_i) \leq k|S| < 2^m$$

□

Lemma 15. $\forall S \subset \{0,1\}^m, |S| > (1-2^{-n})2^m, \exists u_1, \cdots, u_k$ such that

$$\cup_{i=1}^k (S+u_i) = \{0,1\}^m$$

Proof. We proof that if $u_1, \cdots, u_k$ being chosen independently and randomly, then $\texttt{Pr}[\cup_{i=1}^k (S+u_i) = \{0,1\}^m] > 0$. In fact, for $r \in \{0,1\}^m$, define B_r be the event that $r \notin \cup_{i=1}^k (S+u_i)$, and we want to proof $\texttt{Pr}[\exists_{r \in \{0,1\}^m} B_r] < 1$, which can be implied if we have $\forall r, \texttt{Pr}[B_r] < 2^{-m}$. Since $B_r = \cap_{i\in[k]} B_r^i$ where B_r^i is the event that $r \notin (S+u_i) \Leftrightarrow (r+u_i) \notin S$,$r+u_i$ is a uniform element in $\{0,1\}^m$, therefore in S with probability greater than $1-2^{-n}$. Finally, B_r^i are independent for different i an d hence

$$\texttt{Pr}[B_r] = \texttt{Pr}[B_r^i]^k \leq 2^{nk} < 2^{-m}$$

□

Having these two lemmas, we have $x \in L$ if and only if

$$\exists u_1, \cdots, u_k \in \{0,1\}^m \forall r \in \{0,1,\}^m, r \in \cup_{i=1}^k (S_x + u_i)$$

which means

$$\exists u_1, \cdots, u_k \in \{0,1\}^m, \forall r \in \{0,1\}^m, \bigvee_{i=1}^k M(x, r \oplus u_i) \text{ accepts}$$

Hence $L \in \Sigma_2^p$ and thus proved our original statement. □

We are also interested in asking is there polynomial hierarchy results on randomized complexity classes. Unfortunately, the original diagonalization technique we used is not valid on probabilistic situations. That is because we can not define the property of **BPTIME** semantically rather than probabilistically. Determining whether a PTM has the property "error chance less than $\frac{1}{3}$" is undecidable, while we can easily verify the encoding of a general Non-Deterministic Turing Machine as we did in construction the universal Turing Machine. This also brings the difficulty to find **BPP**-Complete problems as we mentioned above. Such as if try to construct a trivial **BPP**-Complete language L by defining it as all tuples $\langle M, x, 1^t\rangle$ where for input x, M output 1 within t steps with probability at lease $\frac{2}{3}$. This kind of technique is widely used in the deterministic cases to construct complete language for some complexity class. But, although L is obviously **BPP**-Hard, but is not known whether in **BPP** since for $\langle M, x, 1^t\rangle \notin L$ we could have $\texttt{Pr}[M(x) = 1] = \frac{1}{2}$, which is greater than $\frac{1}{3}$.

In conclusion, the relationship between randomized complexity classes and others is still full of sandwiched relations and uncertain inclusions and they could easily collapse only have a few equals between them, just like other area of complexity theories does. There is still a long way to go for a proper classification.

References

[1] Donald E. Knuth. *The TEXbook*, Volume A of Computers and Typesetting, Addison-Wesley, Reading, Massachusetts, 1984.

[2] Michael Sipser. *Introduction to the Theory of Computation*, Thomson, 2006.

[3] C. Papadimitriou. *Computational Complexity*, Addison-Wesley, 1995.

[4] Sanjeev Arora, Boaz Barak. *Computational Complexity - A Mordern Approach*, Cambridge, 2007.

[5] O. Carton. *Langages Formels, Calculabilite et Complexite*, Vuibert, 2008.

[6] Manindra Agrawal, Neeraj Kayal, Nitin Saxena. *PRIME is in P*, Annals of Mathematics, 2004.

[7] Thomas H. Cormen, Charles E. Leiseison, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithm*, MIT Press, 2001.