

Probabilistic Computations
and
Complexity Classes
A Survey

By: Rave Harpaz *

May 21, 2002

*Computer Science Department, The Graduate Center,
City University of New York.

Outline:

- NP- a probabilistic characterization
- Quantifier notations of complexity classes
- Randomized Computations
 - Definitions
 - RP and coRP (randomized poly-time)
 - ZPP (zero-error probabilistic poly-time)
 - BPP (bounded-error probabilistic poly-time)
 - PP (probabilistic poly-time)
 - Summary and open problems
- Interactive Proof Systems
 - Introduction
 - Model description
 - Definitions
 - GNI- Graph Non-Isomorphism
 - The power of IP
 - Public-coin Vs. Private-coin, Arthur-Merlin games
- PCP- Probabilistic Checkable Proofs
 - Development
 - Definitions
 - The PCP theorem

Definition 1. NP

$NP = \{L \mid \text{there exists a polynomial time NDTM that decides } L\}$

Definition 2. NP (an alternative probabilistic definition)

$L \in NP \Leftrightarrow \exists M$ (poly-time NDTM) s.t. $\forall x$:

$$x \in L \rightarrow Pr[M \text{ accepts } x] > 0$$

$$x \notin L \Rightarrow Pr[M \text{ rejects } x] = 1$$

Definition 3.

$coNP = \{\bar{L} \mid L \in NP\}$

As opposed to NP, only "no" instances (disqualifications) of problems in coNP possess short proofs (certificates).

Examples:

VALIDITY, HAMILTON PATH COMPLEMENT $\in coNP$

Some results:

- $L \in NP\text{-complete} \Rightarrow \bar{L} \in coNP\text{-complete}$
- $P = NP \Rightarrow NP = coNP$
- $L \in coNP\text{-complete} \wedge L \in NP \Rightarrow NP = coNP$
- $L \in NP \cap coNP \Rightarrow L \notin NP\text{-complete}$ (unless $NP = coNP$)

Quantifier notations of complexity classes

This notion will not only provide a compact way to characterize complexity classes, but also yields a uniform treatment which leads to easier proofs for class-inclusion and hierarchy-collapse results.

In addition to the familiar length-bounded quantifiers $(\exists y, |y| = k)$ and $(\forall y, |y| = k)$ we define $(\exists^+ y, |y| = k)$ to mean, "For most strings y such that $|y| = k$," and $(\P y, |y| = k)$ to mean "For over half of the strings y such that $|y| = k$,".

We call \exists^+ the "overwhelming majority" quantifier, and \P the "majority quantifier". The difference is that the threshold for \P is "just over" 50% whereas for \exists^+ it is bounded away from 50% by a fixed amount.

Using these quantifiers we can give alternative definitions to many probabilistic complexity classes, e.g. let $P(\dots)$ be a polynomial-time predicate, and $p(\cdot)$ a polynomial. Then $L \in NP$ iff there exists some $P(\dots)$ and $p(\cdot)$ such that for all x ,

$$\begin{aligned} x \in L &\rightarrow (\exists y, |y| = p(|x|)) P(x, y) \\ x \notin L &\rightarrow (\forall y, |y| = p(|x|)) \neg P(x, y) \end{aligned}$$

Knowing that our quantifiers are polynomially bounded we can rewrite our definitions, e.g.

$$L \in NP \Leftrightarrow (\forall x) : x \in L \rightarrow \exists y P(x, y) \quad \text{and} \quad x \notin L \rightarrow \forall y \neg P(x, y)$$

We can further abbreviate this by writing: $NP = (\exists/\forall)$

Another example, recall that:

$$\begin{aligned} L \in \Sigma_2 \Leftrightarrow (\forall x) : x \in L \rightarrow \exists y_1, \forall y_2 P(x, y_1, y_2) \quad \text{and} \\ x \notin L \rightarrow \forall y_1 \exists y_2 \neg P(x, y_1, y_2) \end{aligned}$$

which can be abbreviated as: $\Sigma_2 = (\exists\forall/\forall\exists)$

Randomized Computations

A **Probabilistic algorithm** is an algorithm designed to use the outcome of a random process, typically it would contain an instruction to "**flip a coin**" which would influence the algorithm's subsequent execution and output. Certain problems are more easily solvable by probabilistic algorithms than by deterministic ones.

Definition 4. probabilistic Turing Machine

Is a type of a nondeterministic Turing machine where each step is called a coin-flip step and has two legal next moves. We assign a probability to each branch b of the machine's computation as follows, where k is the number of coin-flips:

$$Pr[b] = 2^{-k}$$

The probability that the machine accepts input w :

$$Pr[M \text{ accepts } w] = \sum_{b \text{ is an accepting branch}} Pr[b]$$

The probability that the machine rejects input w :

$$Pr[M \text{ rejects } w] = 1 - Pr[M \text{ accepts } w]$$

Definition 5. Las Vegas and Monte Carlo Machines

A **Las Vegas** machine will always give the correct answer when it terminates. A **Monte Carlo** machine may sometimes produce an incorrect solution (though we are always able to bound the probability of incorrect solutions). There are two kinds of Monte Carlo machines: a **one-sided error** machine in which it may err on only one of the yes-no answers, and a **two-sided error** machine in which it may err on both yes-no answers. Note that a Las Vegas machine is a Monte Carlo machine with error probability 0.

The classes **RP** And **coRP**

Randomized polynomial time, one-sided error computable languages.

Definition 6. RP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M , such that

$$x \in L \Rightarrow \Pr[M \text{ accepts } x] \geq \frac{1}{2}$$

$$x \notin L \Rightarrow \Pr[M \text{ rejects } x] = 1$$

(i.e. no false positives, accept by majority and reject unanimously)

Definition 7. coRP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M , such that

$$x \in L \Rightarrow \Pr[M \text{ accepts } x] = 1$$

$$x \notin L \Rightarrow \Pr[M \text{ rejects } x] \geq \frac{1}{2}$$

(i.e. no false negatives, accept unanimously and reject by majority)

An alternative way to define coRP is:

$$\text{coRP} = \{L | \bar{L} \in \text{RP}\}$$

Definition 8. RP and coRP (quantifier definition)

$L \in RP \Leftrightarrow (\forall x) : x \in L \rightarrow \exists^+ y P(x, y) \quad \text{and} \quad x \notin L \rightarrow \forall y \neg P(x, y)$

Abbreviated: $RP = (\exists^+/\forall) \quad coRP = (\forall/\exists^+)$

Theorem 1. $P \subseteq RP, P \subseteq coRP, RP \subseteq NP, coRP \subseteq coNP$

proof: A polynomial deterministic Turing machine is a special case of an RP machine, it is one that ignores the coin-flips and also accepts unanimously (probability 1 is at least as large as $\frac{1}{2}$).

An RP machine is by definition a nondeterministic machine, since if half of the computations accept, surely at least one does.

Invariance of the constant

The constant $\frac{1}{2}$ in the definition of RP is arbitrary. We could have chosen any constant ϵ (between 0-1) and still get the same complexity class. If the constant is less than $\frac{1}{2}$ we can, by an **amplification** procedure bring the threshold to a level $\geq \frac{1}{2}$. The amplification procedure runs the RP machine a certain number of times (depending on the constant) and returns "yes" if one of the runs returns "yes". If none of the runs return a "yes" then it is obvious that the probability of getting an incorrect answer falls, thus increasing the probability of getting a correct answer.

It can even be shown that we can replace the constant ϵ by $\frac{1}{p(|x|)}$ or $1 - 2^{-p(|x|)}$, which are thresholds that depend on the length of the input.

Note that by employing this amplification procedure we can obtain the characterization: $RP = (\exists^+/\forall) = (\frac{1}{p}/\forall)$

The class **ZPP**

zero-error probabilistic polynomial time, modeling Las Vegas computations.

Definition 9. ZPP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M containing an additional final state called "don't know", such that

$$x \in L \Rightarrow \Pr[M \text{ accepts } x] > \frac{1}{2} \wedge \Pr[M \text{ rejects } x] = 0$$

$$x \notin L \Rightarrow \Pr[M \text{ rejects } x] > \frac{1}{2} \wedge \Pr[M \text{ accepts } x] = 0$$

(i.e. no false positives, no false negatives, but we may get a "don't know" answer)

Again the value $\frac{1}{2}$ is arbitrary and can be replaced as before with any value between $2^{-p(|x|)}$ and $1 - \frac{1}{p(|x|)}$.

Theorem 2. $ZPP = RP \cap coRP$

Proof: *To show that $ZPP \subseteq RP$ we label the "don't know" states in the ZPP machine with "reject" to obtain an RP machine deciding the same language. Similarly to show that $ZPP \subseteq coRP$ we label "don't know" states with "accept". Thus $ZPP \subseteq RP \cap coRP$. To show that $RP \cap coRP \subseteq ZPP$ we construct a ZPP machine from the RP and coRP machines (M_1, M_2) as follows:*

*run M_1 , if it accepts then accept
run M_2 , if it rejects then reject
otherwise return "don't know"*

Note that a quantifier form of definition for ZPP is not known.

Theorem 3. $P \subseteq ZPP$

Since $P \subseteq RP \cap coRP$.

The class **BPP**

bounded-error probabilistic polynomial time, two-sided error computable languages.

Definition 10. BPP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M , such that

$$x \in L \Rightarrow Pr[M \text{ accepts } x] \geq \frac{1}{2} + \epsilon$$

$$x \notin L \Rightarrow Pr[M \text{ rejects } x] \geq \frac{1}{2} + \epsilon$$

(i.e. accept by "clear majority" and reject by "clear majority")

The BPP machine makes mistakes but returns the correct answer most of the time. By running the machine a large number of times and returning the majority of the answers we can guarantee that the probability of a mistake is very low. Although the amplification process is less trivial as before we can still build BPP machines with thresholds ranging from $\frac{1}{2} + \frac{1}{p(|x|)}$ to $1 - 2^{-p(|x|)}$.

Even though BPP machines can have exponentially small error probabilities, no such machine is known to solve a decision problem, not solvable by more constrained machines.

Definition 11. BPP (quantifier definition)

$L \in BPP \Leftrightarrow (\forall x) : x \in L \rightarrow \exists^+ y P(x, y)$ and $x \notin L \rightarrow \exists^+ y \neg P(x, y)$
Abbreviated: $BPP = (\exists^+/\exists^+) = (\forall/\exists^+)$

Theorem 4. $RP \cup coRP \subseteq BPP$

$RP \subseteq BPP$ and $coRP \subseteq BPP$ because a one-sided error machine is a special case of a two-sided error machine.

Theorem 5. $coBPP = BPP$

This is because the definition of a BPP machine is symmetric.

The class **PP**

probabilistic polynomial time, two-sided error computable languages.

Definition 12. PP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine M , such that

$$x \in L \Rightarrow Pr[M \text{ accepts } x] > \frac{1}{2}$$
$$x \notin L \Rightarrow Pr[M \text{ rejects } x] \geq \frac{1}{2}$$

(i.e. accept by "regular majority" and reject by "regular majority")

The PP class is wider than what we have seen so far. In the BPP case we had a gap between the number of accepting and rejecting computations, wide enough to enable us to invoke the machine polynomially many times to notice the difference between inputs that are in the language and those that are not. PP machines do not put the gap restriction which may be very small, hence amplification to a desired level can not always be achieved in polynomial time. Therefore languages in PP have no realistic computational content, similar to NP.

Definition 13. PP (quantifier definition)

$L \in PP \Leftrightarrow (\forall x) : x \in L \rightarrow \exists y P(x, y) \quad \text{and} \quad x \notin L \rightarrow \exists y \neg P(x, y)$

Abbreviated: $PP = (\exists/\forall)$

Theorem 6. $BPP \subseteq PP$

Trivial, because $\frac{1}{2} + \epsilon > \frac{1}{2}$.

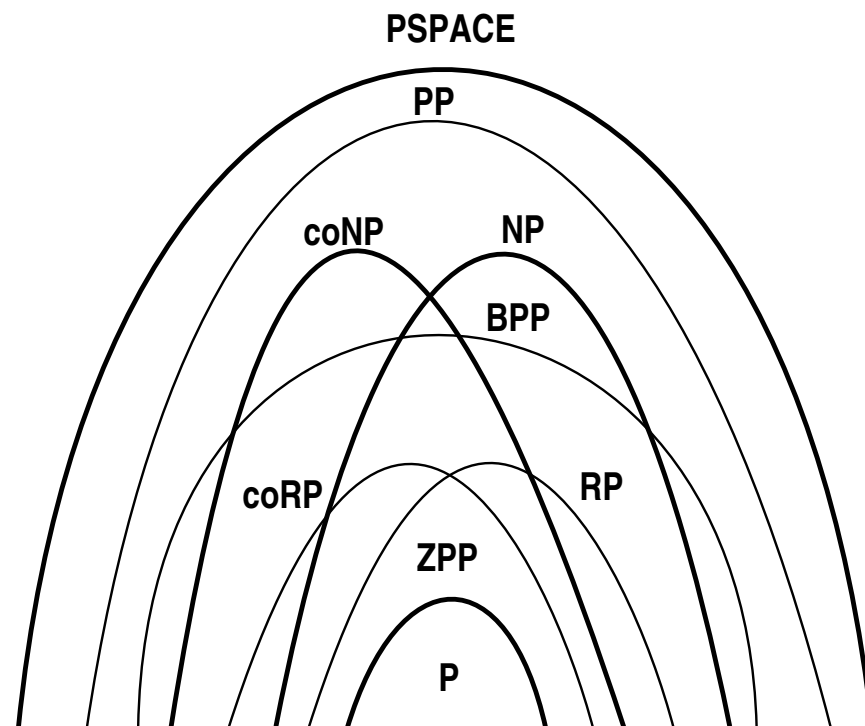
Theorem 7. $PP \subseteq PSPACE$

Run all possible computation paths with length $p(|x|)$ reusing the space and count the number of accepting and rejecting states.

Theorem 8. $NP \cup coNP \subseteq PP$

Combine the two machines M_1, M_2 deciding $L_1 \in NP, L_2 \in coNP$ into one machine by adding a new start state.

Summary



We Know:

- $P \subseteq ZPP$
- $ZPP = RP \cap coRP$
- $RP \cup coRP \subseteq BPP$
- $RP \subseteq NP$ and $coRP \subseteq coNP$
- $BPP \subseteq PP$
- $PP \subseteq PSPACE$

Open problems:

- $P = NP?$
- $NP = coNP?$
- $P = NP \cap coNP?$
- $PRIMES \in P?$
- $RP = coRP?$
- $RP = NP \cap coNP?$
- $BPP \subseteq NP?$

It is believed that $BPP = P$, hence the following hierarchy $P \subseteq ZPP \subseteq RP \subseteq BPP$ will collapse and randomized computations will be of no real use.

Interactive Proof Systems

- Introduced by Goldwasser, Micali and Rackoff for cryptographic applications (1989), and independently by Babai as a game theoretic extension of NP(1985).
- Provide a probabilistic analog of the class NP , similar to the way probabilistic algorithms provide a probabilistic analog of the class P .
- In this context a proof is viewed as a process rather than a fixed object. It is viewed as way in which a *prover* convinces a *verifier* of a certain claim.
- In an interactive proof system the focus is on the task of the *verifier*- the verification process, which is typically much easier than finding a proof.

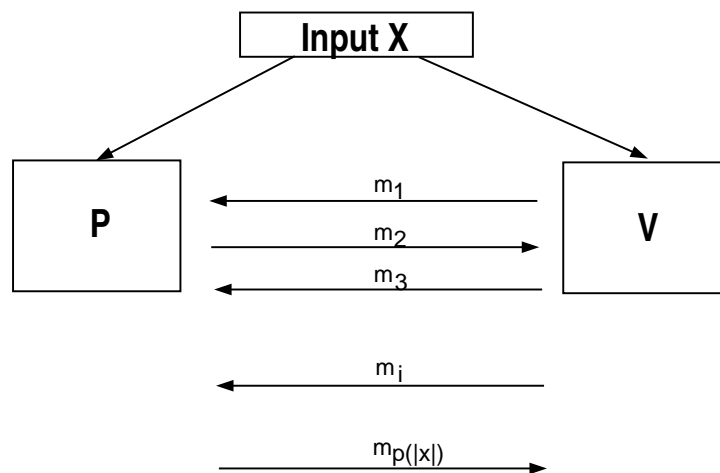
NP as a proof system

The languages in NP are those whose members all have short certificates of membership that can easily be checked. This can be rephrased as a proof system in which a proof for a membership claim such as " $x \in L$ " (where L is an NP language) consists of the prover sending a witness " y ", and the verifier checking in polynomial-time whether " y " is indeed a certificate for " x ". Note that the prover is computationally unbounded.

take the SAT problem for example, the prover can convince a polynomial-time verifier that a formula ϕ is satisfiable simply by supplying the satisfying assignment.

Description of the model

1. The verifier V is a deterministic polynomial-time Turing machine.
2. The prover P is computationally an unlimited (in space and time) deterministic Turing machine.
3. An interactive proof is a sequence of questions and answers between the P and V . V asks P questions, P based on the history of the dialog and the current question, answers back.
4. At the end of the interaction, V decides based on the knowledge he acquired during the process whether a claim P is trying to prove is true or false.
5. Both the length and the number of messages exchanged by P and V are bounded by a polynomial in the length of the input.



Definition 14. A language L has an interactive proof system- (P, V) if there exists a verifier V such that:

$\exists P$ where (P, V) accepts all $x \in L$ (Completeness)

$\forall P', (P', V)$ rejects all $x \notin L$ (Soundness)

The first requirement (Completeness) states that if $x \in L$ then there is always a way to prove ($\exists P$) this fact to V . The second requirement (Soundness) states if $x \notin L$ then there is no way to prove the contrary to V , i.e. it is not possible to prove a false theorem. The later requirement is motivated by the fact that we do not want V to trust P , i.e. even if P is providing incorrect answers, V will be able to detect it.

By extending our model to allow V to be a polynomial-time probabilistic Turing machine (requirement 1), we can define a new very large class called IP .

Definition 15. IP

The class of all languages L for which there exists a probabilistic polynomial-time Turing machine V , such that:

$$x \in L \Rightarrow \exists P, Pr[(P, V) \text{ accepts } x] \geq \frac{2}{3}$$

$$x \notin L \Rightarrow \forall P', Pr[(P', V) \text{ accepts } x] \leq \frac{1}{3}$$

The first requirement states that for any $x \in L$ there exists a prover that can convince the verifier that $x \in L$ with high probability. The second requirement states if $x \notin L$ then there is no prover that can convince the verifier that $x \in L$ with better than low probability.

By an amplification process similar to BPP , the probabilities $\frac{2}{3}$ and $\frac{1}{3}$ can be replaced by probabilities as extreme as $1 - 2^{-p(|x|)}$ and $2^{-p(|x|)}$, i.e. making the error probability exponentially small.

Example- Graph Non-Isomorphism (GNI)

Definition 16. Graph Isomorphism (GI)

instance: two graphs G_1, G_2 where $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$

question: are G_1, G_2 isomorphic ($G_1 \cong G_2$)? i.e. does there exist a bijection $\pi : V_1 \rightarrow V_2$ such that $(u, v) \in E_1 \Leftrightarrow (\pi(u), \pi(v)) \in E_2$.

GI is an important and odd problem, it is obviously in NP , but is not known to be NPC , or in P (or even in $co-NP$, or BPP). Accordingly its complement Graph Non-Isomorphism is not known to be in NP , BPP or P .

Definition 17. $GNI = \{(G_1, G_2) | G_1 \not\cong G_2\}$

Theorem 9. $GNI \in IP$

Verifier's algorithm

```
begin {input :  $G_1, G_2$ }  
  succ:=true;  
  repeat twice  
  begin  
     $i := \text{random}(1, 2)$ ;  
    randomly create a graph  $H$  isomorphic to  $G_i$ ;  
    send  $H$  to prover;  
    receive  $j$  from prover;  
    if  $i \neq j$  then succ:= false;  
  end; if succ then accept else reject;  
end.
```

Prover's algorithm

```
begin {input :  $G_1, G_2$ }  
  receive  $H$  from verifier;  
  if  $H$  is isomorphic to  $G_1$  then send 1 else send 2 to verifier;  
end.
```

In case $G_1 \not\cong G_2$ the prover will always send the correct j , because H can be isomorphic to at most one of G_1 or G_2 . In case $G_1 \cong G_2$ the prover has no way of knowing whether H was generated from G_1 or G_2 , and can only answer randomly. In such a case the probability of sending the correct j to the verifier is $\frac{1}{2}$. Repeating the process twice results a probability of $\frac{1}{4}$ that the prover will successfully convince the verifier that $G_1 \not\cong G_2$.

Theorem 10. $NP \subseteq IP$

Proof: *By allowing interaction only between the prover and verifier, i.e. disabling the probabilistic facility of the verifier, making it a deterministic poly-time machine. the prover computes a membership proof and sends it to the verifier, which can check it in poly-time.*

Theorem 11. $BPP \subseteq IP$

Proof: *By allowing the probabilistic facility of the verifier only and no interaction between the prover and verifier. The prover sends a witness and the verifier runs a BPP algorithm to check its validity.*

Note that by allowing the probabilistic facility of the verifier only we obtain the class $IP(1)$ which is also denoted MA (Merlin-Arthur), which seems to be a randomized (and perhaps stronger) version of NP .

The power of IP

For some years the relations between IP and other complexity classes were not clear. It was believed that IP is slightly larger than NP and it was conjectured that $coNP$ complete problems did not belong to IP .

More evidence, in the following result suggests that proving that $coNP$ complete problems do belong to IP would be a hard task.

Theorem 12. $\exists A$ s.t $coNP^A - IP^A \neq \emptyset$

This result shows that there exists a separator language that separates the relativized versions of $coNP$ and IP , thus proving or disproving that $coNP \neq IP$ will require proof techniques that do not relativize, i.e. digitalization and simulation techniques will not suffice.

Theorem 13. $coNP \subseteq IP$

The proof consists of deriving an interactive proof for $\overline{3SAT}$, by arithmetization of the 3 – CNF formula and applying algebraic methods related to prime numbers.

Theorem 14. $IP = PSPACE$ (Shamir)

Proving that $IP \subseteq PSPACE$ consists of simulating in polynomial space the interaction between the prover and the verifier and determining its outcome. This is done by enumerating over all the random choices of the verifier and accepting only if more than $\frac{2}{3}$ of the outcomes accept.

Proving that $PSPACE \subseteq IP$ consists of constructing an interactive proof for the $PSPACE$ -complete problem $TQBF$ and novel algebraic methods of analyzing computation, similar to the methods used to prove $coNP \subseteq IP$.

Public-coin Vs. Private-coin

In a private-coin interactive proof system the verifier does not reveal his random choices to the prover. This corresponds to our definition of IP .

Definition 18. *Public-coin interactive proofs- AM (Arthur-Merlin)* Arthur-Merlin games are a special case of the interactive proof systems, where Arthur plays the role of the verifier and merlin the all powerful magician plays the role of the prover. The only difference is that now the verifier reveals his random choices. In each round the verifier can now only send the prover the outcome of is random choices.

Denote $AM = AM(2)$, i.e. a proof system in which at most 2 round are used.

Question: Are the two approaches equivalent ?

Theorem 15. $\forall r(n) \quad IP(r(n)) \subseteq AM(r(n) + 2)$

the following theorem shows that $AM(r(n))$ is invariant under a linear change in the number of rounds (linear speed-up)

Theorem 16. $\forall r(n) \geq 2 \quad AM(2r(n)) = AM(r(n))$

Corollary 1. $\forall r(n) \geq 2 \quad IP(2r(n)) = IP(r(n))$

Corollary 2. $IP(O(1)) = AM(2)$

Probabilistic Checkable Proofs (PCP)

Development

- The concept of PCPs evolved out of an extension of interactive proofs; *multi-prover interactive proofs (MIP)*, introduced by Ben-Or, Goldwasser, Kilian and Wigderson (1988).
- An *MIP* model consists of a random polynomial-time verifier communicating with two infinitely powerful provers who cannot communicate with each other. A language L is in *MIP* if there exists a verifier V that is always convinced when $x \in L$, but when $x \notin L$ the provers have only a small probability of convincing V of the contrary.
- An equivalent model of *MIP* was suggested by Fortnow, Rompel and Sipser (1988). In this model the verifier is interacting with an Oracle that is trying to convince him that $x \in L$. Since oracle's replies are fixed in advance, we can think of an oracle as a string of bits representing a proof that $x \in L$, to which the verifier has random access. Hence, *MIP* can be characterized as the set of languages for which a membership proof can be verified probabilistically in polynomial time.
- After showing that $IP = PSPACE$, Babai, Fortnow and Lund showed that $MIP = NEXPTIME$. This result, combined with the oracle formulation of *MIP* shows that if a language has membership proofs of exponential size, then some probabilistic verifier can check these proofs.
- The next step was "scaling-down", i.e. showing efficient verification procedures for checking membership proofs of smaller nondeterministic classes, such as NP .

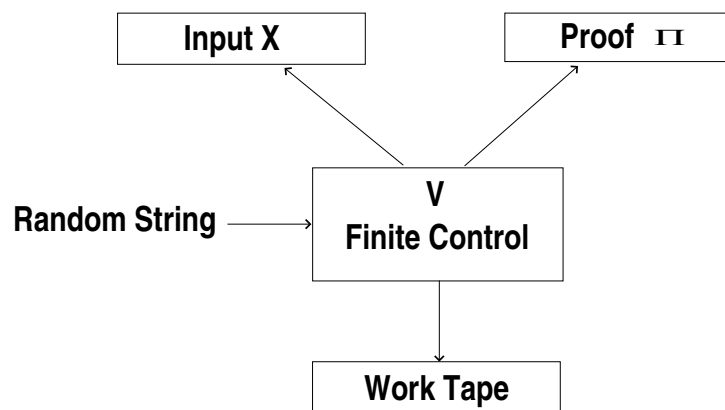
- Another approach was to scale down just the number of *random* bits and *query* bits required to check a proof, instead of scaling down the running time of the verifier.
- Arora and Safra (1992) formalized the class PCP and used it to give the class NP a new probabilistic definition.
- With *PCP* we examine how robust are oracle proofs with respect to local perturbation, that is, how many parts of a proof can be changed while still maintaining the proof recognizable as a correct one, and how much randomness we need to check the correctness of a proof.
- One of the main contributions of *PCP* is to the area of approximation algorithms, in terms of negative results, i.e. how hard will it be to approximate certain problems.
- A main inapproximability result due to Feige et al. showed that if any poly-time algorithm can achieve a constant approximation ratio for MAX-CLIQUE then all NP problems are solvable in $n^{O(\log \log n)}$.

The PCP theorem

Definition 19. A **verifier** is a polynomial-time probabilistic Turing Machine- M , used to verify membership proofs. It contains:

- *Input tape*
- *Work tape*
- *A source of random bits, called the **random string**, denoted- τ*
- *read only tape called the **proof string**, denoted- Π*

M has random access to Π . The work tape contains a special addressing portion on which M can write the address of a location in Π , and then read just the bit in that location. This operation is called a **query**.



Definition 20. A verifier is $(r(n), q(n))$ – restricted if on each input of size n it uses at most $O(r(n))$ random bits for its computation, and queries at most $O(q(n))$ bits of the proof.

M operates as follows: on input x , where $|x| = n$, it reads τ of length $cr(n)$, computes a sequence of $kq(n)$ locations and queries those locations in Π . Depending on what these bits were, it accepts or rejects.

Definition 21. $M^\Pi(x, \tau) = 1$ if M accepts input x , with access to proof Π , using τ .

Definition 22. A verifier M can probabilistically check membership proofs for language L if:

- $x \in L \Rightarrow \exists \Pi$ s.t. $Pr[M^\Pi(x, \tau) = 1] = 1$
(i.e. for every random string, M accepts with probability 1).
- $x \notin L \Rightarrow \forall \Pi$ $Pr[M^\Pi(x, \tau) = 1] < \frac{1}{2}$
(i.e. M rejects all proofs with probability at least $\frac{1}{2}$).

Definition 23. The complexity class $PCP(r(n), q(n))$ is a class consisting of all languages L for which membership proofs can be checked by a $(r(n), q(n))$ – restricted verifier.

Note that $NP = PCP(0, poly(n))$, since $PCP(0, poly(n))$ is the class of languages for which membership proofs can be checked in deterministic polynomial-time, without randomness.

An alternative way to characterize NP in terms of PCP is the following theorem.

Theorem 17 (PCP Theorem). $NP = PCP(\log n, 1)$

References

1. S. Zachos. Probabilistic Quantifiers and Games. *Journal of Computer and System Sciences*, 36(3):433-451, June 1988.
2. S. Arora and M Safra. Probabilistic Checking of Proofs: A New Characterization of NP. *Journal of ACM*, 45(1):70–122, 1998.
3. C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley,1994.
4. D.P.Bovet and P.Crescenzi. *Introduction to the Theory of Complexity*. Prentice hall, 1994.
5. O. Goldreich. *Introduction to Complexity Theory*. Lecture notes.
6. M. Sipser. *Introduction to the Theory of Computation*. PWS, 1997.