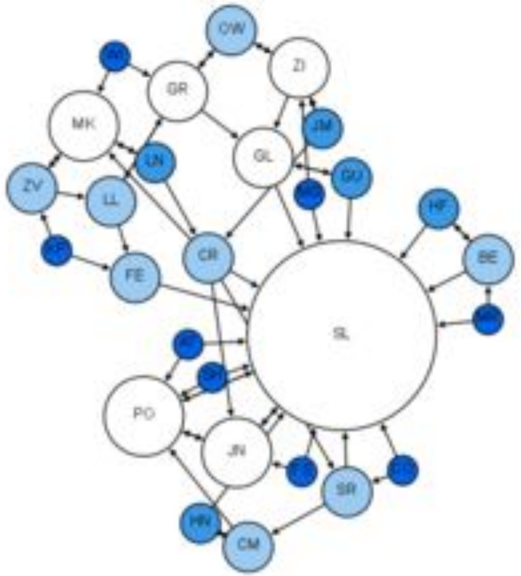
SAPIENZA
UNIVERSITÀ DI ROMA

On the Complexity of Approximation and Combinatorial Optimization

Senad Beadini



Why Approximation ?

- Suppose somebody wants to convince you that a Boolean formula F is satisfiable. He could present the usual certificate, namely, a satisfying assignment, which we could then check by substituting back into the formula.

However, doing this requires reading and checking the entire certificate.

I’ll present a complexity class that allows to overcome this issue.
(probabilistically checkable proof system (PCP))

Definitions :

<u>Def</u> 1 : A language *L* is in NP if there is a poly-time Turing machine V ("verifier") that, given input *x*, checks certificates (or membership proofs) to the effect that x ∈ L. This means :

$$
\begin{aligned}
&x \in L \Rightarrow \exists \pi \text{ s.t. } V^{\pi}(x) = 1\\
&x \notin L \Rightarrow \forall \pi \quad V^{\pi}(x) = 0,
\end{aligned}
$$

where Vπ denotes "a verifier with access to certificate π".

- The class **PCP** (short for "Probabilistically Checkable Proofs") is a generalization of this notion. Therefore PCP is also a generalization of NP !!!

Definitions 2 :

<u>**Def**</u> **2 :** Let *L* be a language and *q*, r : N → N. We say that *L* has an **(*r*(n), *q*(n))-verifier** if there's a polynomial-time probabilistic TM *V* satisfying:

1) <u>Efficiency:</u> On input x ∈ {0, 1}^n and given random access to a string π ∈ {0, 1}*, V uses at most r(n) random coins tosses (choices) and makes at most q(n) queries to locations of π. The output is "1" ("accept") or "0" ("reject").

2) <u>Completeness:</u> If x ∈ *L* then there exists a proof π ∈ {0, 1}* such that :

$$\text{PROB} [V\pi (x) = 1] = 1.$$

We call π the correct proof for x.

3) <u>Soundness:</u> If x not in L then for every proof π ∈ {0, 1}* we have that

$$\text{PROB} [V\pi (x) = 1] \leq 1 / 2$$

<u>**Def**</u> **3 :** We say that a language L is in **PCP(r(n), q(n))** if *L* has a (O(r(n)), O(q(n)))-verifier.

Notes :

- In this framework the verifier is probabilistic.

- The verifier has random access to the certificate string π. This means that each bit of the certificate can be independently queried by the verifier via a special address tape: if the verifier desires say the i-th bit in the proof string, it writes i on the address tape and then receives the bit π[i].

  (This is equivalent considering the Verifier as a oracle TM with oracle π.)

- A generalization of NP. Instead of conducting a polynomial-time computation upon receiving the entire proof (as in the case of NP), here the verifier is allowed to toss coins and query the proof only at locations of his choice.

- The values r(n) and q(n) strongly characterized PCP.
  q(n) determines what is the portion of the proof being read by the verifier
  r(n) the number of coins tossed by the TM determines what is the total number of possible executions.

- Proofs checkable by an (r, q)-verifier contain at most q * 2^r bits

The verifier looks at only q places of the proof for any particular choice of its random coins, and there are only 2^r such choices.

- Note that an oracle can possibly be of exponential length (since one can specify an exponentially far location on the string using polynomially many bits).

PCP Theorem : A big result

$$\mathbf{NP} = \mathbf{PCP}(\log n, 1)$$

- NP is the class of languages having a PCP system whose verifier makes a **constant** number of queries while using a logarithmic number of coin tosses !!!
- Very powerful theorem.. huge implications in combinatorial optimization and many others fields..
- One of the most complicated proofs in complexity theory [1]. It's far beyond the scope of this presentation.
- I will prove a weaker result : $\mathrm{PCP}(log, poly) \subseteq \mathrm{NP}$

[1] Arora, Sanjeev, and Shmuel Safra. "Probabilistic checking of proofs: A new characterization of NP." *Journal of the ACM (JACM)* 45.1 (1998): 70-122.

Theorem : $\mathrm{PCP}(log, poly) \subseteq \mathrm{NP}$

I have to prove that : if L ∈ PCP(log, poly) ⇒ L ∈ NP

So I'll show how to use its PCP system in order to construct a non-deterministic Turing machine **M'** which decides L in polynomial time.

Proof :

- Let **M** be the probabilistic polynomial oracle machine in the above PCP(log, poly) system for L
- We are guaranteed that on input x ∈ {0, 1}*, **M** makes poly(|x|) queries using O(log |x|) coin tosses.
- Let's denote by {r_1, . . , r_m} the set containing all possible outcomes of the coin tosses made by **M**.
  (note that | r_i | = O(log(|x|)) and m = 2^O(log(|x|)) = poly(|x|))
- Denote by $(q_1^i, \ldots, q_{n_i}^i)$ the sequence of ni queries made by **M** when using the coin sequence r_i
- By the completeness condition we are guaranteed that for every x ∈ L there exists a PCP proof πx such that the verifier M always accepts x when given access to πx.

Theorem : $\mathrm{PCP}(log, poly) \subseteq \mathrm{NP}$

I have to prove that : if L ∈ PCP(log, poly) ⇒ L ∈ NP

So I'll show how to use its PCP system in order to construct a non-deterministic Turing machine **M'** which decides L in polynomial time.

Proof :

A natural candidate for an NP-witness (certificate) for x would be πx.

⇩

We can't use πx .. πx might be of exponential size in |x| and therefore unsuitable to be used as an NP-witness.

How can we get our poly (in |x|) certificate ?

Use a **compressed** version of **πx**. This version corresponds to the portion of the proof which is actually being read by the PCP verifier **M**.

Compressed version of πx

Consider all possible executions of M on input x given access to the oracle string π (each execution depends on the coin sequence r_i).

We can take the substring of π containing all the bits πq_i_j examined by M during these executions. Encode each entry in this substring as (index, π_index) (that is, (query, answer)), and denote the resulting encoded string by wπ x (note that now |wπx | is polynomial in |x|).

Let's come back to the theorem.

Theorem : $\mathrm{PCP}(log, poly) \subseteq \mathrm{NP}$

I have to prove that : if $L \in \mathrm{PCP}(\log, \mathrm{poly}) \Rightarrow L \in \mathrm{NP}$

So I'll show how to use its PCP system in order to construct a non-deterministic Turing machine **M'** which decides L in polynomial time.

Proof :

- Let's build now our machine NTM **M'**

NTM M' :

1) On input x
2) Guess non-deterministically the part of the oracle needed by M. Let this guess be called w.
3) Simulates the execution of **M** on input x for all possible r_i (possible executions).
   - Since we have w, **M'** knows all the answers to the queries that **M** can make to the proof/oracle.
   - **M'** will accept if and only if M would have accepted x for all possible r_i 's

Complexity? Since **M'** simulates the execution of M exactly m = 2^O(log(|x|)) (which is polynomial in |x|), and since **M** is a polynomial time machine, **M'** itself is a polynomial time machine, as required.

The last step :

It remains to see that L(**M’**) indeed equals L(**M**) !!

1) For all x ∈ L, we show that there exists a w such that **M’** accepts x. By the completeness condition of the PCP system for L, there exists an oracle π such that Pr [**Mπ** accepts x] = 1. Therefore, it holds that for all coin sequences r_i, **M** accepts x after accessing π.

⇩

It immediately follows by definition that **M’** accepts when guessing Wπx.

2) For all x ∉ L, we show that for all w’s it holds that **M’** with input x always rejects. Indeed by the soundness condition of the PCP system for L, for all oracles π it holds that Pr [**Mπ** accepts x] ≤ 1/2.

⇩

Therefore, for every π, there is at least one coin sequence r_i for which **M** does not accept x when accessing **π**. Hence, for every Wπx guessed by **M’** there must also be at least one coin sequence r_i for which **M’** reaches a rejecting state for M, which means that **M** has rejected x.

∎

Discussion : Why PCP theorem is so interesting ?

- **The PCP Theorem implies that for many NP optimization problems, computing near-optimal solutions is no easier than computing exact solutions.**

Example :

MAX 3SAT is the problem of finding, given a 3CNF Boolean formula ϕ as input, an assignment that *maximizes* the number of satisfied clauses. This problem is NP-hard, since the corresponding decision problem, 3SAT, is NP-complete

For every 3CNF formula ϕ, we define $\mathrm{val}(\phi)$ as the maximum fraction of clauses that can be satisfied by any assignment to ϕ's variables. In particular, if ϕ is satisfiable then $\mathrm{val}(\phi) = 1$.

Let $\rho \leq 1$. An algorithm A is a ρ-approximation algorithm for MAX 3SAT if for every 3CNF formula ϕ with m clauses, $A(\phi)$ outputs an assignment satisfying at least $\rho \cdot \mathrm{val}(\phi)m$ of ϕ's clauses.

Theorem by Hastad said that for every $\rho > 0$, if there is a polynomial-time ($7/8 + \rho$)-approximation algorithm for MAX 3SAT then P = NP.

Corollary : For every $\rho > 0$, computing ($7/8 + \rho$)-approximation to MAX 3SAT is NP-hard.

Other relation :

- PCP is related with CSP (Constraint Satisfaction Problems) problems.

Theorem (Equivalent to PCP theorem) :

There exist constants $q \in N$, $\rho \in (0, 1)$ such that ρ-gap qCSP is NP-hard.

What is a ρ-GAP problem?

Example with 3SAT : Let $\rho \in (0, 1)$. The ρ-GAP 3SAT problem is to determine, given a 3CNF formula ϕ whether:

- ϕ is satisfiable, in which case we say ϕ is a YES instance of ρ-GAP 3SAT.
- $\mathrm{val}(\phi) \leq \rho$, in which case we say ϕ is a NO instance of ρ-GAP 3SAT

3CNF formulas are a special case of 3CSP instances.

Conclusion :

- PCP theorem expresses that NP is the class of languages having a PCP system whose verifier makes a **constant** number of queries while using a logarithmic number of probabilistic choices.

- Approximating the optimal solution for certain problems is not easier than compute the optimal solution.

- In literature there are different versions of the theorem involving directly approximation algorithms like Max3SAT but in general CSP.

- PCP theorem implies that approximate the *optimal* solution of a *general* optimization problem CSP is NP-hard.

A more practical example is a non-convex optimization problem. That is in general NP-hard.

Thank you for your attention