Course "Proofs and Computers", JASS'06

Probabilistically Checkable Proofs

Lukas Bulwahn

May 21, 2006

Contents

1 Introduction to Probabilistically Checkable Proofs

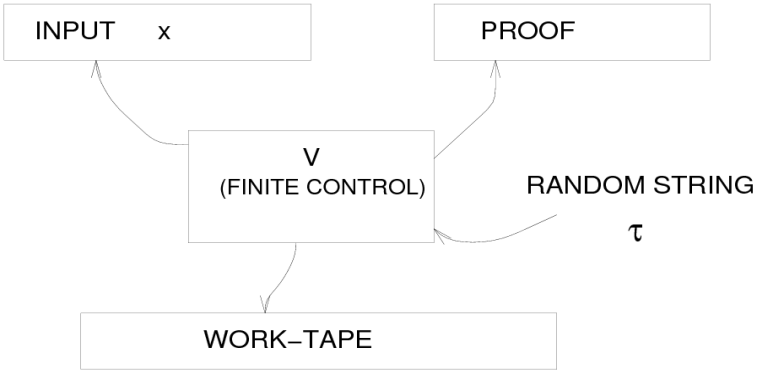
1.1 History of Inapproximability Results

Before introducing probabilistically checkable proofs, I shortly give an overview of the historical development in the field of inapproximability results which are closely related to PCPs.
A foundational paper from Johnson in 1974 states approximation algorithms and inapproximability results for Max SAT, Set Cover, Independent Set, and Coloring.
While the decision problems for various problems, such as Max SAT, Set Cover, Independent Set, and Coloring, were shown to be NP-hard by Cook, Levin and Karp, it was difficult to show approximability and inapproximability results with the known reductions.
To prove inapproximability results, there was a new model for NP necessary, this evolved from works on multi-provers interactive proofs from Ben-Or and Goldwasser.
In 1991, Feige and Goldwasser created this new model and showed new inapproximability results with this model. This new model was later on called PCP.
In 1992, Arora [**?**] could prove the PCP Theorem which made PCP easier and useful to apply for many inapproximability problems.
Since then, many computer scientists could prove and improve many inapproximability results creating tight results for many NP-hard problems.
In 2005, Dinur has published a new proof for the PCP Theorem. This will be introduced in the paper from Bernhard Vesenmeyer, the necessary tools for this proof will be introduced in the sections about constraint graphs and expander graphs.

1.2 A (r,q)-restricted verifier

First, we have to define the verifier which will prove for an input x to be part of the language or not.

Definition 1. A verifier V is a *(r,q)-restricted verifier* if for any input x, witness w, and random string τ of length $O(r)$, the decision $V^w(x,\tau) = "yes"$ is based on at most $O(q)$ bits from the witness w.



This verifier can also be seen as an interactive proof system, but in this interactive proof system the number of random bits of the verifier is restricted.
In the definition of the (r,q)-restricted verifier, you see that we will consider the number of random bits we can use (randomness complexity) and the number of queries to the witness (query complexity). Specifying the random and query complexity results in different classes for languages as we will see in the examples later.
A (r,q)-restricted verifier is called non-adaptive if the queries to the witness w only depend on the input x and the random string τ. If the next queries are also dependant from the previous queries from the witness w, the verifier is called adaptive.
For the sake of simplicity we will consider a verifier non-adaptive from now on.

1.3 Probabilistically checkable proofs

Definition 2. A language L is *probabilistically checkable* using an (r,q)-restricted verifier V iff

- Completeness: If $x \in L$ then there exists a witness w such that $Pr_\tau[V^w(x,\tau) = "yes"] = 1$.
- Soundness: If $x \notin L$ then for every witness w we have $Pr_\tau[V^w(x,\tau) = "yes"] < 1/2$.

Comparing this with the definition of interactive proof systems, we also see the two-sided error is reduced to an one-sided error up to $\frac{1}{2}$ for an input which

is assumed to be not accepted. It is clear that the boundary of $\frac{1}{2}$ is mostly arbitary, as you can run many (up to any constant c) proofs which can reduce the probability of failure to any $\epsilon > 0$. However, the number of queries is still interesting, since for inapproximability results this creates the specific gap of problem. Today, it is known that 11 bits are enough and there are still efforts to lower the number.

Example 3. Some simple examples for PCP-Classes are:

- P = PCP(0, 0) : This verifier has no witness and no randomness. Therefore it can only work on the input x, just like a deterministic polynomial-time bound Turing machine.
- NP = PCP(0, poly) : This verifier has no randomness and a witness which can be read at polynomial many positions, this is identical to a NP witness. Without the randomness, acceptance is inprobabilistic. The verifier is the same as a non-deterministic polynomial-time bound Turing machine.
- NP $\subseteq$ PCP(log, poly) : From the example above, it is simple to see that by allowing logarithmical many random bits one will not reduce the computation of the verifier.
- co-RP = PCP(poly, 0) : The verifier has no witness and can use random bits in each step. The probability of false positive is less than 1/2. This equals the definition of $co - RP$.

These examples are simple because at least one of the complexities is set to zero. More interesting is how increasing one of the complexities and decreasing the other changes the power of the verifier. This will finally lead to the PCP Theorem.

2 PCP Theorem - Part 1

Contents

First notice that it can be easily shown that PCP(log,poly) $\subseteq$ NP. So it is clear that NP = PCP(log,poly).
The amazing fact is now that it can also be shown that:

2.1 PCP Theorem

Definition 4 (PCP Theorem)**.** $NP = PCP(log(n), 1)$

The PCP Theorem states that the verifier only has to look at a constant number of bits of the witness string in order to probabilistically say if the input is in the language or not.

First, we will prove the easier side of the PCP theorem: PCP(log(n), 1) $\subseteq$ NP. The main idea of the following proof is that a non-deterministic Turing machine V' can create all random strings with logarithmic length of a verifier V and simulate the calculation with the input x and all created random strings in polynomial time. The first step is to explain how the (log(n), 1)-verifier V works with the input x.

Proof.
Let L $\in$ PCP(log(n), 1) $\Rightarrow$ there is a (log(n), 1)-verifier V. For τ there are $2^{O(log(n))} \leq n^c$ many random strings, namely $\tau^1, \cdots, \tau^{n^c}$. The verifier V will work as follows:

1. Reads a random string $\tau^i, 1 \leq \ i \leq n^c$.
2. Uses x and τ^i to calculate q positions $i_1, \cdots, i_q$ to read from the witness string.
3. Run a calculation with x and $w_{i_1}, \cdots, w_{i_q}$, and answer "yes" or "no".

Proof.
Now, we will simulate the verifier V on a non-deterministic Turing machine V'. The witness string for V' is w which has polynomial length since V can only access polynomial positions. V' now calculates step 2 and 3 from V for every possible τ^i and answers "yes" if all simulated calculations of V answered "yes".

It is left to show that V' behaves like V.

Proof.
- $x \in L$ and $L \in PCP(log(n), 1) \Rightarrow$ For a given w, V returns yes with probability 1. With this witness w V' will also return yes.
- $x \notin L \Rightarrow$ There is no witness string w for V' because at least on half of the calculations will not answer "yes".

□

As the verifier V' can simulates all possible random runs of V, one would think at first glance that NP is more powerful than PCP(log(n), 1). Showing the inclusion of NP $\subseteq$ PCP(log(n), 1) is one of the most difficult proofs in theory of computation - an overview of the original proof can be found in [**?**] - and therefore we will first show how to apply the PCP Theorem brings great results in inapproximability.

3 Applying PCP Theorem

3.1 Selected parts of approximability

Contents

Simply speaking, an approximation problem is an problem which one is not interested in finding the best possible solution, but in finding a solution which

is close to best solution and its computation is still efficient.
The following definitions will formally describe the problem:

Definition 5. An *optimization problem* O is defined by a cost function $C : \Sigma^* \times \Sigma^* \to R_+ \cup \{\perp\}$, that given an instance string x and a solution string s outputs $C(x, s)$ which is either the cost of the solution or $\perp$ if the solution is illegal. Let $OPT(x)$ denote the optimal value a solution can get, then: $OPT(x) = max_{s:C(x,s)\neq\perp} C(x, s)$. An optimization problem is to find a legal solution s^* that attains the optimal value of cost, $C(x, s^*) = OPT(x)$.

Example 6. MAX-3SAT is the problem of finding an assignment A which maximizes the percent of satisfied clauses of a 3CNF formula ψ. Of course, if ψ is satisfiable, then the optimal value of MAX-3SAT is 1.

Definition 7. A is an *r-approximation algorithm* for a maximation problem iff for any input x, A finds a solution s that $C(x, s) \geq rOPT(x)$.

In this paper, we are using the definition that $0 < r < 1$ and the better an approximation is, the higher r is.
In other literature, you also can find the definition for $r' = 1/r$ and $r' > 1$, then the better an approximation is, the lower r' is.
It is clear that reducing a decision problem is easier than reducing an r-approximation problem, so we introduce the definition for gap problems:

Definition 8. Let O be a maximization problem. Let x be an instance of the problem. A *gap(α, β)-O* is the problem of deciding between the following alternatives:

- "Yes": $OPT(x) \geq \beta$
- "No": $OPT(x) \leq \alpha$

If $OPT \in [\alpha, \beta)$ then both alternatives are acceptable.

Intuitively, A gap problem is to split all possible inputs into the ones for which the optimal solution is less than α and the others for which the optimal solution is greater than β. If the gap problem is NP-hard, then it is the $\frac{\alpha}{\beta}$-approximation algorithm is also NP-hard. Now, reductions from gap problems can be used instead of working directly with r-approximation problems.

3.2 Equivalence of PCP Theorem and gap-MAX-3SAT is NP-hard.

Contents

In this section, we will reduce gap-MAX-3SAT to 3SAT using the PCP Theorem.
There can also be found many other reductions using the PCP Theorem, e.g.

for the clique number or the chromatic number of graphs in [**?**].

Lemma 9. *The following statements are equivalent:*

1. *(PCP Theorem)* $NP = PCP(log(n), 1)$
2. *There exists* $\alpha \in (0, 1)$*, such that gap(*α*, 1)-MAX-3SAT is NP hard.*

Proof.
(2 $\Rightarrow$ 1) Let language $L \in NP$. Assumption: gap(α, 1)-MAX- 3SAT is NP hard. $\Longrightarrow$ there exists a 3CNF formula, $\psi_{x,L} = c_1 \wedge \ldots \wedge c_m$, such that

1. $x \in L \Leftrightarrow \psi_{x,L}$ is satisfiable.
2. $x \notin L \Leftrightarrow$ for every assignment A, the number of clauses in $\psi_{x,L}$ that are satisfied is less than αm.

The verifier V can use the following algorithm to check if a string x is in the language L:

Algorithm

1. step: Construct the 3CNF formula $\psi_{x,L}$.
2. step: Get an assignment A and create witness/proof $w = \psi_{x,L} \circ A$.
3. step: Choose $k = O(1)$ clauses from the witness.
4. step: If all k clauses are satisfied, return "yes".

Proof.
- Completeness: If the assignment A satisfies the formula, V will answer "yes" no matter which k clauses were chosen.$\surd$
- Soundness: If A does not satisfy $\psi_{x,L}$, then it satisfies at most αm clauses $\Longrightarrow$the probability to answer "yes" is at most α^k. With $k > log(1/2)/log(\alpha)$: $Pr_\tau[V^w x, \tau = "yes"] \leq \alpha^k \Longrightarrow Pr_\tau[V^w(x, \tau) = "yes"] < 1/2.\surd$

□

Proof.
(1 $\Rightarrow$ 2) Proof by reduction from gap-MAX-3SAT to 3SAT. $3SAT \in NP \Longrightarrow$ $3SAT \in PCP[log, 1] \Longrightarrow$ there exists a verifier V such that a given 3CNF formula ϕ:

- ϕ is satisfiable $\Rightarrow \exists w : Pr_\tau[V^w(\phi, \tau) = "yes"] = 1.$
- ϕ is not satisfiable $\Rightarrow \forall w : Pr_\tau[V^w(\phi, \tau) = "yes"] < 1/2.$

Proof.
The verifier only considers q bits of the witness w for its decision. $\Rightarrow$ acceptance is determined with local constraint ψ_τ^ϕ and variable assignment according to the positions in the witness w. It is still true that:

- ϕ is satisfiable $\Rightarrow$ all local constraints ψ_τ^ϕ are satisfied.
- ϕ is not satisfiable $\Rightarrow$ for any assignment A at most half of the local constraints are satisfied.

Proof.
Construct a new formula $\phi' = \psi_{\tau_1}^\phi \wedge \cdots \wedge \psi_{\tau_n^c}^\phi$ with $\tau_1, \cdots \tau_{n^c}$ are all random string of the length $O(log(n))$. For ϕ', we have:

- ϕ is satisfiable $\Rightarrow$ ϕ' is satisfied.
- ϕ is not satisfiable $\Rightarrow$ any assignment for ϕ' satisfies at most half of the clauses of ϕ'.

Proof. Construct a 3CNF formula from each local constraint. $\phi'' = \underbrace{(\psi_{1,1} \wedge \ldots \wedge \psi_{1,k})}_{\psi_{\tau_1}^\phi} \wedge \ldots \wedge \underbrace{(\psi_{n^c,1} \wedge \ldots \wedge \psi_{n^c,k})}_{\psi_{\tau_{n^c}}^\phi}$

- ϕ is satisfiable $\Rightarrow$ ϕ'' is satisfied.
- ϕ is not satisfiable $\Rightarrow$ any assignment for ϕ'' satisfies at most $\frac{2k-1}{2k}$ of the clauses of ϕ''.

This concludes the reduction from gap(α,1)-MAX-3SAT for $\alpha = \frac{2k-1}{2k}$ to 3SAT assuming the PCP Theorem. □

3.3 Outlook of MAX-3SAT

After we have shown that there exists an α, it would be interesting for which α this was already shown.

Theorem 10 (John Hastad, 1997)**.** *For any $\alpha \in \left(\frac{7}{8}, 1\right)$, the problem gap($\alpha$,1)-MAX-3SAT is NP-hard.*

The proof for this theorem can be found in [**?**].

Fact
But notice this interesting fact: Howard Karloff and Uri Zwick have stated a $\frac{7}{8}$-Approximation Algorithm for MAX-3-SAT and provided *strong evidence* that the algorithm performs equally well on arbitrary MAX-3-SAT instances.

So for this problem, there has been found a best possible performing algorithm. The approximation algorithm and the inapproximability result have created a clear boundary for MAX-3-SAT. For other problems, research are still going on. To become an overview of the results, one can find tables in [**?**], [**?**] and [**?**].

4 PCP Theorem - Part 2

The following presentation [**?**] will present the proof for the PCP Theorem by Gap Amplification following the proof of Irit Dinur [**?**]. This proof uses constraint graphs and expander graphs. Therefore, we shortly will introduce these two topics.

4.1 Constraint Graphs

Contents

Definition 11. $G = \langle (V, E), \Sigma, C \rangle$ is called a *constraint graph*, if

1. (V,E) is an undirected graph, called the underlying graph of G.
2. The set V is also viewed as a set of variables assuming values over alphabet Σ.
3. Each edge $e \in E$ carries a constraint $c^e : \Sigma^2 \to \{T, F\}$ and $C = \{c^e\}_{e \in E}$.

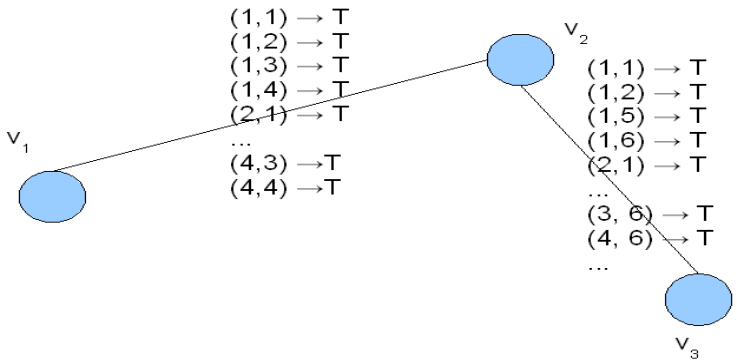
Definition 12. An *assignment* is a mapping $\sigma : V \to \Sigma$ that gives each vertex in V a value from Σ. For any assignment σ, define $SAT_\sigma(G) = Pr(c^e(\sigma(u), \sigma(v)) = T)$ and $SAT(G) = max_\sigma SAT_\sigma(G)$.

Example 13. Constructing a constraint graph from a 3-SAT-formula: $\phi = \underbrace{(A \vee B \vee C)}_{v_1} \wedge \underbrace{(A \vee D \vee E)}_{v_2} \wedge \underbrace{(D \vee F \vee G)}_{v_3}$

1. Encode each clause as a vertex.
2. Encode the satisfying assigments to a clause as the alphabet Σ.

(T, T, T)	(T, T, F)	(T, F, T)	(T, F, F)	(F, T, T)	(F, T, F)	(F, F, T)
1	2	3	4	5	6	7

3. Put a consistency constraint for every pair of clauses that a share a variable.



Theorem 14. *Given a constraint graph $G = \langle (V, E), \Sigma, C \rangle$ with $|\Sigma| \leq 7$, it is NP-hard to decide if $SAT(G) = 1$.*

Proof. Use the example from above to reduce to 3SAT. As 3SAT is NP-hard to decide, it is NP-hard to decide if $SAT(G) = 1$. □

To make it short, one can say a constraint graph is another data structure for boolean SAT-formulas.

4.2 Expander Graphs

Contents

Simply speaking, an expander graph is a graph which *expands quickly*, which means that any subset of vertices is connected to many vertices of the complement set. This means expander graphs are graphs which are *quite much* random. To get these two sentences more formally, we will introduce a few definitions and a few examples.

Definition 15. Let $G = (V, E)$ be a d-regular graph. Let $E(S, \overline{S}) = \left|(S \times \overline{S}) \cap E\right|$ equal the number of edges from an non-empty subset $S \subseteq V$ to its complement. The edge expansion is defined as $h(G) = min_{S,|S|\leq|V|/2} \frac{E(S,\overline{S})}{|S|}$.

To get an intuitive sight of the edge expansion, we should look at the following simple examples:

Example 16.
- A disconnected graph has an expansion of 0.

  Proof. As the graph is disconnected, you can choose a connected component of the graph whose size is less than $|V|/2$ for S. Now $E(S, \overline{S}) = 0$, and therefore $h(G) = 0$. □

- A random d-regular graph has an expansion of about $d/2$, independent of the number of vertices.

  Proof. Let S be a subset of at most n/2 vertices of a random d-regular graph. A typical of vertex in S is connected to $d \cdot frac|\overline{S}|n$ vertices in $\overline{S}$. So $\frac{E(S,\overline{S})}{|S|} \approx d \cdot frac|\overline{S}|n$. As $|\overline{S}|$ is minimal at about $n/2$, $h(G) \approx d/2$. □

Lemma 17. *There exist $d_0 \in N$ and $h_0 > 0$, such that there is a polynomial-time constructible family $\{X_n\}_{n\in N}$ of d_0-regular graphs X_n on n vertices with $h(X_n) \geq h_0$.*

Instead of proving this lemma, we are instead presenting the following example for a family of expander graphs.

Example 18. All graphs of size p (for all primes). Here $V_p = Z_p$ and $d = 3$. Ever vertex is connected to its neighbors ($x + 1$,$x - 1$) and its inverse (x^{-1}).

Working with expander graphs, one is especially interested in eigenvalues of the adjacency matrix of an expander G. These eigenvalues are also the Spectrum of the graph G.
Some simple observations are that the largest eigenvalue of an expander is d or that isomorphic graphs have the same spectrum.
Simply speaking, the largest eigenvalue tells you to how many vertices are connected to each other and the second largest eigenvalue tells you something about which vertices are connected to each other. This surely is very vague, but the next two definitions show what is meant exactly.

Lemma 19. *Let G be a d-regular graph, $h(G)$ denotes the edge expansion of G and let $\lambda(G)$ be the second largest eigenvalue of the adjacency matrix of G. Then $\lambda(G) \leq d - \frac{h(G)^2}{d}$.*

This lemma shows the relation between the edge expansion h and the second largest eigenvalue.
The following lemma shows how the second largest eigenvalue tells you something about the way vertices are connected.

Lemma 20 (Expander Mixing Lemma)**.** *for all $S, T \subseteq V$:* $\left|E(S,T) - \frac{d|S||T|}{n}\right| \leq \lambda\sqrt{|S|\,|T|}$

If you look at the lemma, $\frac{d|S||T|}{n}$ is exactly the number of edges you expect that connect S and T in a typical random graph. And $E(S,T)$ are the number of edges that connect S and T in the given graph. So the difference of those two terms stands for the difference of the given graph G to a typical random graph, this is obviously bound by λ. So, a small λ means a graph with allot of "randomness".

Theorem 21. *Let $G = (V, E)$ be a d-regular graph with a second largest eigenvalue λ. Let $F \subseteq E$ be a set of edges. The probability p that a random walk that starts at a random edge in F takes the $i + 1$st step in F as well, is bounded by $\frac{|F|}{|E|} + \left(\frac{\lambda}{d}\right)^i$.*

Example 22 (Amplifying the success probability of random algorithms). $L \in RP$. A decides whether $x \in L$ with m coin tosses and one-sided-error probability β. Simple way: Pr(A fails)$\leq \beta^t$ and uses $m \cdot t$ coin tosses. With random walk on expander graphs: Pr(A fails)$\leq (\beta + \frac{\lambda}{d})^t$ and uses $m + t \cdot log(d)$ coin tosses.

4.3 Probability

Contents

Here we will shortly present a lemma from probability which will be used in the following presentation.

Lemma 23. *For any non-negative variable X, $Pr(X > 0) \geq \frac{E^2(X)}{E(X^2)}$.*

Proof. X is non-negative $\Longrightarrow E(X^2) = E(X^2|X > 0) \cdot Pr(X > 0)$ and $E(X) = E(X|X > 0) \cdot Pr(X > 0)$. $\Longrightarrow \frac{E^2(X)}{E(X^2)} = \frac{(E(X|X>0) \cdot Pr(X>0))^2}{E(X^2|X>0) \cdot Pr(X>0)} \leq Pr(X > 0)$. because $E(X^2|X > 0) \geq E^2(X|X > 0)$. □

4.4 Conclusion

As this paper was written for JASS'06, I suggest to also read the paper [**?**] to get the complete picture of Probabilistically checkable proofs.

References

[Aro94] Sanjeev Arora. *Probabilistic Checking of Proofs and Hardness of Approximation Problems*. PhD thesis, CS Division, UC Berkeley, 1994.

[Bel96] Mihir Bellare. Proof checking and approximation: Towards tight results. *Sigact News, Vol.27, No. 1*, 1996.

[Din04] Irit Dinur. Advanced topics in complexity: Pcp theory, 2004.

[Din05] Irit Dinur. The pcp theorem by gap amplification. *Electronic Colloquium on Computational Complexity*, 2005.

[Has97] John Hastad. Some optimal inapproximability results. In *the 28th Annual ACM Symposium on Theory of Computing*, 1997.

[HK97] Uri Zwick Howard Karloff. A 7/8-approximation algorithm for max 3sat?, 1997.

[NL03] Avi Wigderson Nati Linial. Expander graphs and their application, 2003.

[Pap94] Christos Papadimitriou. *Computational complexity*. Addision-Wesley, 1994.

[SH95] A. Steger S. Hougardy, H.J. Prömel. Probabilistically checkable proofs and their consequences for approximation algorithms. *Trends in Discrete Mathematics*, 1995.

[Ves06] Bernhard Vesenmeyer. The pcp theorem by gap amplification, 2006.

[Weg05] Ingo Wegener. *Complexity theory : exploring the limits of efficient algorithms*. Springer, 2005.