Technion IIT
Department of Computer Science
Fall 2007-8

Course 236603:

Probabilistically Checkable Proofs

Eli Ben-Sasson

Table of Contents

Homework assignments

References

Lecture 1

Statement of the PCP Theorem

January 20th, 2008

Lecturer: Eli Ben-Sasson Scribe: Eli Ben-Sasson

This lecture is devoted to defining formally what a *probabilistically checkable proof* (PCP) is and stating a couple of useful variants of the PCP Theorem. In this and the next lecture we will give examples of applications of (the two variants of) the PCP Theorem and show why this theorem is justly considered to be one of the great achievements of theoretical computer science in the past couple of decades.

1.1 Trading certainty for computational efficiency

What constitutes a mathematical proof? Without going into a discussion of mathematical logic, let us describe a few important properties of formal mathematical proofs. The first and most important property is that proofs can be checked automatically by a *machine*. While finding a mathematical proof may require the elusive properties known as "intelligence", "creativity" and "ingenuity", it is well-known since the beginning of the 20^{th} century that checking a proof, if properly written, requires no intelligence at all. Proofs can be encoded formally as a sequence of bits and there exists an algorithm that decides whether a string of bits is an encoding of a legal proof of a mathematical statement encoded by another string of bits. In the language of theoretical computer science, we can say that the language consisting of all true mathematical statements that are implied by a set of axioms (a *theory*, to use the terminology of mathematical logic), is decided by a nondeterministic Turing machine. The nondeterministic choices of our machine on an input statement correspond to the sequence of bits that encodes a formal proof.

A crucial property that is perhaps overlooked in courses on mathematical logic, is that the machine that verifies proofs is *computationally efficient*. If one inspects any of the standard formalizations of mathematics, as presented in any introductory book to mathematical logic, one finds that the algorithm that checks whether a string of bits forms a legal proof, runs in polynomial time in the length of the proof. In fact, in most cases the running time is close to linear in the length of the proof. Two other properties that are crucial to the definition of a proof-verifying nondeterministic machine are its *completeness* and *soundness*. Completeness means that every true statement has a proof and soundness means that every statement that is not true (or is ill-formulated) has no proof.

The PCP Theorem says that the computational efficiency of the proof checking machine and be greatly increased. For instance, one well-known variant of the theorem (Theorem 1.3) says that the machine can check proofs by relying on only three randomly selected bits of the proof. The improved efficiency comes with a price. Our efficient *verifier* must rely on

random coin tosses and it may err by either rejecting correct proofs of true statements or by accepting pseudoproofs of false statements. However, the probability of error in the verifier's decision depends only on the number of bits it reads and not on the length of the statement or the proof. Moreover, the probability of error decreases exponentially with the number of bits read from the proof, so if we are willing to tolerate a small error probability (say, 2^{-50}) we can check proofs by reading a small constant number of bits from them (150 bits in the case of error probability 2^{-50}). Another price incurred by using a computationally efficient verifier is that the proof needs to be written in a special format that facilitates its efficient verification. The conversion of a "classical" proof into a *probabilistically checkable* one increases the length of the proof and requires extra computation on the part of the party writing down the proof. Fortunately, the conversion can be performed in polynomial time and the resulting proof length can be made *quasilinear*, i.e., classical proofs of length n are transformed into probabilistically checkable ones of length $n\text{polylog}(n)$. This length-efficient variant of the PCP Theorem is stated as Theorem 1.4. Next, we give a formal definition of the class of languages decided by PCP verifiers.

1.2 Complexity Classes defined by PCP verifiers

At the core of a PCP system lies a *verifier* — the randomized machine that verifier proofs of statements. A proof π is usually viewed as a sequence of ℓ symbols from some finite alphabet Σ. However, since we will severely limit the number of symbols read from a proof, we prefer to view it as an *oracle*, i.e., as a function $\pi : \{1, \ldots, \ell\} \to \Sigma$.

Definition 1.1 (PCP-Verifier)**.** A PCP-verifier, or simply, verifier, is a randomized Turing machine, denoted V, with access to an oracle which is called a *proof oracle*, or simply, proof and is denoted by π. On input x and random coin tosses $R \in \{0,1\}^*$, V makes a number of queries to π and outputs either accept or reject. We denote by $V^\pi[x;R]$ the output of V on input x, proof π and random coins R.

Being interested in efficient verifiers, we are going to limit some of their computational resources such as the running time, the number of bits read from the proof and the length of the proof. Additionally, we will require that the verifier make a correct decision with sufficient probability. Good proofs of correct statements must be accepted with a minimal probability called the *completeness* parameter. Purported proofs of incorrect statements will be rejected with a minimal probability known as the *soundness* parameter. The probability of error in both the completeness and soundness cases depend on the random coin tosses of the verifier. Both the restrictions and the completeness and soundness parameters may depend on the length of the input statement that needs to be proved. Once the limitations on computational resources are placed and the allowed error probabilities are defined we have effectively defined a complexity class. Any language that can be decided with the specified certainty probabilities by a resource-limited verifier belongs to this class. The formal definition follows.

Definition 1.2 (PCP Class)**.** Given a list of computational restrictions, a completeness function $c : \mathbb{N}^+ \to [0,1]$ and a soundness function $s : \mathbb{N}^+ \to [0,1]$, the complexity class $\mathbf{PCP}\left(\begin{array}{c|c} \text{list of restrictions} & \begin{array}{c} c(n) \\ s(n) \end{array} \end{array}\right)$ includes all languages $L \subseteq \Sigma^*$ that satisfy the following conditions.

- **Operation:** L has a verifier V operating under the listed restrictions and for every $x \in \Sigma^*, |x| = n$ the following holds.
- **Completeness:** If $x \in L$ there exists a proof π such that

$$\Pr_R[V^\pi[x;R] = \mathsf{accept}] \geq c(n).$$

- **Soundness:** If $x \notin L$ then for every proof π,

$$\Pr_R[V^\pi[x;R] = \mathsf{reject}] \geq s(n).$$

1.3 Two variants of the PCP Theorem

We next present two variants of the PCP Theorem. The first achieves a nearly optimal tradeoff between the amount of information read from the proof and the certainty parameters of the proof. The proof needed for such a process is of polynomial length and the actual polynomial is quite large. The second variant is very efficient in terms of the length of the proof, however, the soundness of this theorem is far from optimal.

Theorem 1.3 (PCP Theorem — query efficient and sound)**.** *For every proper complexity function $f : \mathbb{N}^+ \to \mathbb{N}^+$ and all $\epsilon > 0$,*

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} q & \leq & 3 & & & \\ \Sigma & = & \{0,1\} & & & \\ \mathsf{nonadaptive} & & & c & \geq & 1-\epsilon \\ \mathsf{query-type} & & XOR & s & \geq & \frac{1}{2}-\epsilon \\ t(n), \ell(n) & \leq & \mathrm{poly}(f(n)) & & & \\ r(n) & \leq & \log(\ell(n)) + O(1) & & & \end{array}\right).$$

Where

- *q denotes the number of queries V makes to the proof oracle.*
- *Σ denotes the alphabet of the proof. Each query is answered with a single element from this alphabet.*
- nonadaptive *means that the set of queries made to the proof and the decision process based on the answers given by the oracle depend only on x and the random coins R, and not on answers given by the oracle to previous queries.*

- $\mathsf{query-type}$ *denotes the class of computations performed by the verifier after receiving the query answers. In the case of XOR, the computation depends only on the XOR of the (three) answer bits.*
- $t(n)$ *denotes the running time of* V *as a function of the input length.*
- $\ell(n)$ *is the length of the proof, or, formally, the largest index of a proof-symbol that may be queried by* V *when given input of length* n*.*
- $r(n)$ *is the number of random bits required by* V *on input of length* n*.*

A few remarks about the previous theorem are due. Notice that improving the completeness or soundness seems unlikely (assuming $\mathbf{P} \neq \mathbf{NP}$). If $c = 1$ and all other parameters are left unchanged then $\mathbf{P} = \mathbf{NP}$ because deciding whether a sequence of bits satisfies a collection of XOR constraints is equivalent to solving a system of linear equations over the two-element field and can be done (say, by Gaussian elimination) in polynomial time. Similarly, if s must be less than $1/2$ because a random proof (where each bit is selected by a random coin toss) will be accepted by V with probability $1/2$. The optimality of the soundness and completeness in conjunction with the small query complexity and simplicity of the query type have far reaching implications to our understanding the limitations of approximation algorithms and this will be the topic of our next lecture.
Our next variant of the PCP Theorem given nearly optimal proof length and verifier running time. Thus, it is more tailored for positive applications to efficient checking of proofs and computations. We will give one example of such an application in our next lecture.

Theorem 1.4 (PCP Theorem — short proofs)**.** *There exists an absolute constant* $\epsilon > 0$ *such that for every proper complexity function* $f : \mathbb{N}^+ \to \mathbb{N}^+$*,*

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} \ell(n) & \leq & f(n)\cdot \mathrm{polylog}(f(n)) & & & \\ t(n) & \leq & f(n)\cdot \mathrm{polylog}(f(n)) & & & \\ r(n) & = & \log(\ell(n)) + O(1) & c & = & 1 \\ q & \leq & 2 & s & \geq & \epsilon \\ \Sigma & = & \{0,1,2\} & & & \\ \mathsf{nonadaptive} & & & & & \end{array}\right).$$

The notation for the list of restrictions is the same as in Theorem 1.3.

1.4 Bibliographical notes

The story of the PCP Theorem and the way its proof evolved is quite interesting. An illustrated and entertaining description of this history can be found in O'Donnell [Autumn 2005]. Each of the two variants of the PCP Theorem stated in this lecture rely on several important works. The basic statement of a constant-query PCP characterization of **NP** appeared in Arora et al. [1998] and relies on Arora and Safra [1998]. The application of

the PCP Theorem to efficient program checking (to be discussed in the following lecture) appeared first in Babai et al. [1991] and the implication of the PCP theorem to hardness of approximation was first observed in Feige et al. [1996]. The query efficient PCP variant presented in Theorem 1.3 appeared in Håstad [1997]. It heavily relies on Raz [1998]; Bellare et al. [1998] as well as on the original proof of the PCP theorem of Arora and Safra [1998]; Arora et al. [1998]. The length-efficient PCP variant presented in Theorem 1.4 appeared in Dinur [2007]. It heavily relies on Ben-Sasson and Sudan [2005]; Ben-Sasson et al. [2004]; Dinur and Reingold [2004].

Lecture 2

Implications of the PCP Theorem

January 27th, 2008

Lecturer: Eli Ben-Sasson Scribe: Gadi Aleksandrowicz

2.1 Reminder of last lecture's definitions

In our last lecture we stated two PCP theorems, and in this lecture we shall see some of their implications. First, a reminder: We presented the concept of a "PCP-Verifier" (Definition 1.1) for a nondeterministic language L — a probabilistic Turing machine with oracle access to a "proof" π which decides if an input x belongs to L by reading a small random portion of the "proof". The verifier is subject to many constraints — its running time is limited, the amount of coin tosses it is allowed to make is restricted, the length of the proof it reads is bounded, the exact type of computation is uses to decide is not arbitrary, and most importantly, the number of bits it reads from the proof is small. In spite of all these limitations, the class of problems our poor verifier can decide is quite large and powerful. This is because we allow the verifier to toss coins and are willing to tolerate mistakes in the verifier's decision, as long as they occur with small probability. We estimate the accuracy of the verifier by measures of *completeness* which is the probability that the verifier correctly recognizes that $x \in L$ using a legit π, and *soundness*, the probability that $x \notin L$ is rejected, no matter which π accompanies it.
Later on, in Section 1.3, we stated two PCP theorems, each asserting the existence of a PCP-verifier with a different set of constraints. The first theorem, Theorem 1.3, was characterized by an excellent combination of query complexity (3 bits) and soundness — as close as we wish to $\frac{1}{2}$. However, the price we pay is large (polynomial length) proof size. The second theorem, Theorem 1.4, trades the soundness (which is a small constant) with a reasonable sized proof. In this lecture we will see some of the uses of the two theorems.

2.2 Hardness of approximation

Let us focus on how the first version of the PCP theorem can be used to obtain theoretical results. Namely, that approximating certain problems is NP-hard. First, let us explain what "approximation" formally means by considering the case of maximization approximation.

Definition 2.1 (Maximization approximation)**.** A maximization problem is defined by a function $OPT : X \to \mathbb{N}^+$, where X is the set of inputs.
An algorithm A is called a $\alpha(n)$-approximation to OPT if for all $x \in X$, $|x| = n$ we have $\alpha(n)\, OPT(x) \leq A(x) \leq OPT(x)$.

In order for the definition to make sense we must have $0 \leq \alpha(n) \leq 1$. The closer $\alpha(n)$ is to 1, the better our approximation.
Now we show a specific example for which the first version of the PCP theorem implies that "non trivial" approximation of this problem is not likely to exist.

Example 2.2. MAX3LIN2
The input to the problem is a matrix $M \in \mathbb{F}_2^{m \times n}$ and a vector $b \in \mathbb{F}_2^m$, where $\mathbb{F}_2$ denotes the two-element field. We assume M has at least one nonzero entry in each row, and no more than 3 (hence the "3" in the name of the problem; the "2" comes from $\mathbb{F}_2$). Note that we can think of each row of M as defining an equation.
We denote by $OPT(M, b)$ the maximum number of equations that are simultaneously satisfiable by some vector π. Formally,

$$OPT(M, b) = \max_{\pi \in \mathbb{F}_2^n} |\{i \in [n] \,|M_i \pi = b_i\}| .$$

First we note that there is a simple $\frac{1}{2}$-approximation to this problem - $A(M, b) = \lceil \frac{m}{2} \rceil$.
It is obvious that $\frac{1}{2} OPT(M, b) \leq A(M, b)$ since $OPT(M, b) \leq m$ (one can't satisfy more equations than M contains). To prove that $A(M, b) \leq OPT(M, b)$ we use a probabilistic argument showing that there is π satisfying at least $\lceil \frac{m}{2} \rceil$ of the equations.
Consider the uniform distribution on all $\pi \in \mathbb{F}_2^n$. For each row $i \in [m]$ define a random variable Z_i, such that

$$Z_i \quad = \quad \begin{cases} 1 & M_i \pi = b_i \\ 0 & M_i \pi \neq b_i \end{cases}$$

Now $P[Z_i = 1] = \frac{1}{2}$. To see that, let j be an entry of M_i such that $M_{ij} \neq 0$ (by our assumptions on M, such j always exists). Partition all π into two sets: $T_1 = \{\pi | M_i \pi = b_i\}$ and $T_2 = \{\pi | M_i \pi \neq b_i\}$. We show a bijection between T_1 and T_2. Given $\pi \in T_1$, we map it to π' where $\pi'_k = \pi_k$ for all $k \neq j$, and $\pi'_j = 1 - \pi_j$. Obviously this function is a bijection given that $\pi' \in T_2$; to see this, note that since $\pi \in T_1$ we have

$$\sum_{k \neq j} M_{ik} \pi'_k - b_i = \sum_{k \neq j} M_{ik} \pi_k - b_i = M_{ij} \pi_j = \pi_j \neq 1 - \pi_j \neq \pi'_j$$

And so $\sum_{k=1}^n M_{ik} \pi'_k \neq b_i$, so $\pi' \in T_2$.
Since Z_i is an indicator random variable, we have $E[Z_i] = P[Z_i] = \frac{1}{2}$. Now define $Z = \sum_{i=1}^m Z_i$. Obviously Z is the number of equations satisfied by the random choice of π. From the linearity of expectation we have $E[Z] = E[\sum_{i=1}^m Z_i] = \sum_{i=1}^m E[Z_i] = \sum_{i=1}^m \frac{1}{2} = \frac{m}{2}$, and therefore there exists π for which $Z \geq \frac{m}{2}$, and since Z can only have integer values, $Z \geq \lceil \frac{m}{2} \rceil$.
Hence, A is a $\frac{1}{2}$-approximation for OPT. Can we do any better? It turns out that the PCP Theorem 1.3 implies that the answer to this question is "probably not much better".

Theorem 2.3. *If there exists a polynomial algorithm which is a* $\frac{1}{2}+\delta$*-approximation of* MAX3LIN2*, for any* $1/2 \geq \delta > 0$*, then* $\mathbf{P} = \mathbf{NP}$.

Proof. Assume A is a $\left(\frac{1}{2}+\delta\right)$-approximation for MAX3LIN2. Apply the PCP Theorem 1.3 with $\varepsilon = \frac{\delta}{4}$ for any **NP**-complete language L. We show how to use A in order to decide L in polynomial time.
Given $x \in L$, $|x| = n$, the behavior of the verifier V whose existence is asserted by the PCP theorem can be described by $3LIN2$ system of equations: for each possible coin toss R we define a line in a matrix $M \in \mathbb{F}_2^{r(n)\times l(n)}$, where each nonzero entry represents an index of a bit that is read from the proof. Since V's response is determined by the XOR of the entries read, we set b_i to be the expected result for the randomness i (note that M, b depends on x).
This system of equations can be generated in polynomial time since $r(n), l(n)$ are polynomial. Now we use A to determine an approximation to $OPT(M, b)$.
Assume $x \in L$. Then the PCP theorem shows that there exists $\pi \in \mathbb{F}_2^{l(n)}$ such that $M_i\pi = b_i$ for at least $1-\varepsilon$ of the rows of the matrix - i.e. $1-\frac{\delta}{4}$ rows. Therefore,

$$\begin{aligned} A(M,b) &\geq \left(\frac{1}{2}+\delta\right) OPT(M,b) \geq \left(\frac{1}{2}+\delta\right)\left(1-\frac{\delta}{4}\right) = \\ &= \frac{1}{2}+\delta-\frac{\delta}{8}-\frac{\delta^2}{4} = \frac{1}{2}+\frac{7\delta-2\delta^2}{8} \geq \\ &\geq \frac{1}{2}+\frac{7\delta-2\delta}{8} = \frac{1}{2}+\frac{5\delta}{8} > \frac{1}{2}+\varepsilon \end{aligned}$$

On the other hand, if $x \notin L$:

$$A(M,b) \leq OPT(M,b) \leq \left(1-\left(\frac{1}{2}-\varepsilon\right)\right) = \frac{1}{2}+\varepsilon.$$

Therefore, to decide L simply check (in polynomial time) whether $A(M,b) > \frac{1}{2}+\varepsilon$ or $A(M,b) \leq \frac{1}{2}+\varepsilon$. We conclude that the existence of a polynomial time $(\frac{1}{2}+\delta)$-approximation algorithm for MAX3LIN2 implies $\mathbf{P} = \mathbf{NP}$, as claimed. □

2.3 A positive result

While the first version of the PCP theorem shows us that there is something we cannot do (given $\mathbf{P} \neq \mathbf{NP}$), the second version can be used in a positive manner. Suppose we wish to download a program from a not-too-trusted website, and also suppose there is a way to design "proofs" that a given program is not harmful (actual work on this subject is done in the area of software verification).
Using the second variant of the PCP Theorem stated in the previous lecture, Theorem 1.4, the software developer can create and store a proof for the harmlessness of his program, which would be possible for us to check in a relatively small amount of time and without

need to download the whole proof (which very well might be much bigger than the program) but only a few bits of information.
Of course, to prevent the software developer from cheating us we need a way to commit him to the proof before he starts sending bits from it - there are cryptographic primitives designed specifically for this task.

2.4 Bibliographical notes

As stated in the Lecture 1, the application of the PCP theorem to understanding limitations of approximation algorithms first appeared in Feige et al. [1996]. Theorem 2.3 appeared in Håstad [1997]. The application of the PCP theorem to efficient computation verification appeared initially in Babai et al. [1991]. The addition of cryptographic methods to allow for efficient proof-checking without downloading the whole proof appeared in Kilian [1992] and in Micali [2000].

Lecture 3

Exponential Length PCPs part I — The Hadamard code is Locally Testable

February 4th, 2008

Lecturer: Eli Ben-Sasson

Scribe: Shir Ben-Israel

In this lecture, we are going to start the proof of a weak version of Theorem 1.3. In the version of this lecture, the verifier will query a constant number of bits (larger than 3) and the size of the proof will be superpolynomial. Later on, we will improve various parameters of this construction, most notably, its length and query complexity.
We shall prove the following theorem:

Theorem 3.1. *For every proper complexity function $f : \mathbb{N}^+ \to \mathbb{N}^+$,*

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} \ell(n) & \leq & 2^{f^2(n)} & & & \\ t(n) & \leq & f^2(n) & & & \\ r(n) & = & O(f^2(n)) & c & = & 1 \\ q & = & 16 & s & \geq & \frac{1}{200} \\ \Sigma & = & \{0,1\} & & & \\ \text{nonadaptive} & & & & & \end{array}\right).$$

The notation for the list of restrictions is the same as in Theorem 1.3.

3.1 Encoding proofs via the Hadamard code

To prove this theorem, we have to show that for every language L that has a nondeterministic Turing machine M that decides L and runs in time $f(n)$, we can construct a PCP verifier with the restrictions, completeness and soundness above.
The naive way would be to ask the prover to write in the proof the nondeterministic choices made by M. The problem is that in this case the verifier would have to query every bit in the proof, while we want to create a verifier that reads only a constant number of bits from the proof.
The first step towards resolving the problem is to replace the machine M with a circuit C with size $f(n)$. Its input are $x's$ bits and y - the bits describing the nondeterministic choices made by M. We have that $x \in L$ if and only if there exists y such that $C(x, y) = 1$.
Next we create a variable for every gate in the circuit C, and specify a constraint for each gate that specifies that the output of the gate should match the gate type and the inputs to the gate. For instance, if g_i, the i^{th} gate, is an AND gate with inputs coming from g_j and g_k, we will require $g_i = g_j \wedge g_j$. Clearly $x \in L$ if and only if there is an assignment to the gates of C that satisfies all constraints and such that the very last gate evaluates to 1.

We would like our probabilistically checkable proof to encode an assignment to the gates of C, such that from the encoding our verifier will be able to query a constant number bits and get the value of any sum of a subset of the gates. Later on, we shall use these subsums to verify that the encoded assignment satisfies C.
In particular, our verifier will read the circuit C and the known input x (but recall that the verifier does not know the nondeterministic decisions specified by y). It will check that the proof π is a legal encoding of some string of bits, denoted y'. Then it will check that y' represents an accepting computation of C.
In this lecture, we will discuss the particular encoding that is used to encode the assignment y and underlies our PCP proof.

Definition 3.2 (Error correcting code)**.** An *error correcting code* is an injective function $E_C : \Sigma^k \to \Sigma^n$ (for $k \leq n$). The code's *message length* is k, the *blocklength* is n, the *rate* is $\frac{k}{n}$, the *alphabet* is Σ and the *relative distance* is

$$d = \min_{x \neq x' \in \Sigma^k} \left(\Delta E\left(x\right), E\left(x'\right)\right),$$

where the relative distance between two words is defined to be

$$\Delta\left(y, z\right) = \frac{\left|\{i : y_i \neq z_i\}\right|}{n}.$$

The code C is the set of codewords,

$$C \subseteq \Sigma^n, C = \left\{E\left(x\right) | x \in \Sigma^k\right\}.$$

The family of codes used to prove Theorem 3.1 is defined next.

Definition 3.3 (Hadamard code)**.** The k-dimensional Hadamard code encodes k bits by codewords of length 2^k. The alphabet is the two-element field $\mathbb{F}_2$. Let $\alpha_1, \ldots, \alpha_{2^k}$ be an ordering of $\mathbb{F}_2^k$, then the k-bit message $a = (a_1, ..., a_k) \in F_2^k$ is encoded by the 2^k-bit codeword

$$(\langle a, \alpha_1\rangle, \ldots, \langle a, \alpha_{2^k}\rangle),$$

where $\langle a, b\rangle = \sum_{i=1}^{k} a_i b_i$. Let H_k be the Hadamard code with k dimensions, i.e., $H_k \subset \mathbb{F}_2^{2^k}$ is the set of codewords of the k-dimensional Hadamard code. Let $\textsc{Hadamard} = \{H_k\}_{k \in \mathbb{N}^+}$ denote the family of Hadamard codes.

For example, set $k = 3$ and consider the encoding of the message $a = (101)$ under the lexicographical ordering of elements of $\mathbb{F}_2^3$. The first bit of the coded word will be $\langle 101, 000\rangle = 0$. The second bit of the code will be $\langle 101, 001\rangle = 1$, and so on. In fact, we can construct a matrix G (called the generating matrix of the code) such that given a message a, its codeword is given by $G \cdot a$. In the case of our 3-dimensional example we have

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

3.2 The Hadamard codes are locally testable

We would like the proof to contain a Hadamard codeword, or at least to be close to one. Here we will show that the family of Hadamard codes is *locally testable*, i.e., there exists a *tester* making a few random queries to a purported codeword. Words in the code are accepted by the tester with probability 1 and words that are far from all codewords are rejected with probability proportional to their minimal distance from (a word in) the code. The resemblance of the following pair of definitions to that of a verifier and a PCP complexity class are of course no coincidence, because the local testability of Hadamard codes will play a crucial role in the proof of Theorem 3.1.

Definition 3.4 (Tester)**.** A *tester* for a family of codes of message length k and blocklength $n = n(k)$ is a randomized Turing machine T with oracle access to a word w of size n. The tester receives as input a unary string 1^k denoting the message length. It tosses random coins and uses them to choose some bits to read from the word. Based on the bits read it outputs either accept or reject. We denote by $T^w\left[1^k, R\right]$ the output of T on oracle w and random coins R.

Definition 3.5 (LTC class)**.** Given a list of computational restrictions, completeness function $c : \mathbb{N}^+ \to [0,1]$ and soundness function $s : \mathbb{N}^+ \times [0,1] \to [0,1]$, the complexity class $\mathbf{LTC}\left(\text{ list of restrictions } \middle| \begin{matrix} c(k) \\ s(k,\delta) \end{matrix}\right)$ includes all languages (codes) $L \subseteq \Sigma^*$ that satisfy the following conditions.

- **Operation:** L has a tester T operating under the listed restrictions and for every $x \in \Sigma^*, |x| = n(k)$ the following holds.
- **Completeness:** If $x \in L$ then

$$\Pr_R[T^x[1^k; R] = \mathsf{accept}] \geq c(k).$$

- **Soundness:** If $x \notin L$ and it is δ-far from L then

$$\Pr_R[T^x[1^k; R] = \mathsf{reject}] \geq s(k,\delta).$$

(We say that w is δ-far from C if for each $w' \in C$: $\Delta(w, w') \geq \delta$).

We are ready to state the main theorem of this lecture, namely, that the family of Hadamard codes is locally testable with query complexity 3.

Theorem 3.6.

$$\textsc{Hadamard} \subseteq \mathbf{LTC} \left(\begin{array}{lcl|lcl} t(n) & = & O(n) & & & \\ r(n) & = & 2n & c & = & 1 \\ q & = & 3 & s(\delta) & = & \min\left\{\frac{\delta}{2}, \frac{2}{9}\right\} \\ \Sigma & = & \{0,1\} & & & \end{array} \right).$$

To prove this theorem, we will design a tester for the Hadamard code. The tester runs under the restrictions above and has the completeness and soundness stated there. It will be rather easy to show that the tester accepts every code word and that it runs under the stated restrictions. The hard part will be to prove its soundness. To do this, we will show that if the rejection probability is low, then our oracle is close to a word of the Hadamard code.

Proof. The tester of the code H_n operates as follows:

1. Choose $a, b \in F_2^n$ at random.
2. Read w_a, w_b, w_{a+b}.
3. Accept if and only if $w_a + w_b + w_{a+b} = 0$.

Completeness: If $w \in H_k$ then there exists $m = m_1, \ldots, m_k \in \mathbb{F}_2^k$ such that $w_a = \sum_{i=1}^{n} m_i a_i$. Similarly, $w_b = \sum_{i=1}^{n} m_i b_i$. Finally,
$w_{a+b} = \sum_{i=1}^{n} m_i (a+b)_i = \sum_{i=1}^{n} m_i a_i + \sum_{i=1}^{n} m_i b_i = w_a + w_b$ so the sum of the three bits will be always 0 and the tester will accept with probability 1 the Hadamard codeword w.

Soundness: Given by the following Lemma, whose proof follows.

Lemma 3.7. $Pr_{a,b}[T^w[a,b] = rej] = \varepsilon < \frac{2}{9} \Rightarrow \Delta(w, H) < 2\varepsilon$.

□

Proof of Lemma 3.7. Define the majority codeword $\phi \in F_2^{2^n}$ by

$$\phi_a = \underset{b \in F_2^k}{\text{majority}} \{w_{a+b} - w_b\}.$$

The Lemma follows from the following two statements, discussed next.

1. $\Delta(\phi, w) < 2\varepsilon$ (ϕ is close to w)
2. $\phi \in H_k$

Proof of 1: Let B be the set of bad indices,

$$B = \{a \mid \phi_a \neq w_a\}.$$

Notice that $\Delta(w, \phi) = \Pr_a[a \in B]$ so it suffices to bound the probability of $a \in B$. By assumption, $\varepsilon = \Pr_{a,b}[w_a + w_b \neq w_{a+b}]$. By the rule of conditional probabilities

$$\varepsilon \geq \Pr_a[a \in B] \cdot \Pr_b[w_a \neq w_{a+b} + w + b \mid a \in B] \geq \Pr_a[a \in B] \cdot \frac{1}{2} = \frac{1}{2}\Delta(\phi, w).$$

This concludes the proof of 1. The proof of 2 is left to Homework assignment 2. For a proof, see Ben-Sasson [Fall 2005][Lecture 2]. □

3.3 Bibliographical notes

The exponential length PCP described in Theorem 3.1 was presented in the original proof of the PCP Theorem by Arora et al. [1998]. The local testability of the Hadamard code described in Theorem 3.6 appeared in Blum et al. [1990].

Lecture 4

Exponential Length PCPs part II — Arithmetization

February 11^{th}, 2008

Lecturer: Eli Ben-Sasson Scribe: Noa Elgrabli and Eyal Granit

We wish to continue with our proof of Theorem 3.1. In order to make things simpler, we first provide a layout of the proof :
Given a language $L \in NTIME(f(n))$, our goal is to present a PCP verifier for L that operates under the limitations imposed by Theorem 3.1. Given an input x, the verifier would need to decide whether $x \in L$ or $x \notin L$. The verifier will also have access to a proof oracle, denoted by π, which it will query during it's operation. Note that while x and L are given externally, the format of the proof π can be chosen to be anything of our liking, as long as we can commit ourselves to the completeness and soundness requirements made by the theorem. So our main question is : What should π encode ? Well, since $L \in NTIME(f(n))$, there exists a non deterministic Turing machine M, that can decide on L in time $f(n)$. If indeed $x \in L$, then M has an accepting computation path while operating on x. If on the other hand $x \notin L$, then all the computation paths of M would reject. Thus we would like π to encode an accepting computation path of M for x. Note however that we also want to make only a *constant* number of queries to π. If π would simply encode the series of operations taken by M while operating on x, the proof length would be $f(n)$, and the verifier would be unable to verify that π indeed represents a valid and accepting computation of M for x, without the completeness and soundness arguments being functions of $f(n)$. Our solution is the following : First we reduce L to $CIRCUIT-SAT$ of size $f(n)$, converting M to a boolean circuit ϕ. The accepting computation path of M for x is replaced by an assignment to the gates of ϕ, denoted α, such that the values of the gates represent a valid computation of ϕ on x, and ϕ accepts (outputs **true**). Second, we construct a matrix representation β for the assignment α, for reasons explained later * . Third, we define π to be the Hadamard encoding of β. Since Hadamard codes are locally testable, the verifier can use the local tester we have seen in the last lecture (Theorem 3.6) to check if π is close to a Hadamard code word, and reject immediately if it's not (meaning that π contains "garbage" for our concern). If on the other hand π is close to a Hadamard code word, then the verifier can use properties of Hadamard encoding to receive large amounts of information about α, using only a constant number of queries to π. With this at hand, the verifier would be able to check if α is a valid and accepting computation of ϕ for x, completing it's job and finishing our proof.
The steps of converting the computation path of M to α and β and then to π are called *Arithmetization*, since M is basically reduced to a set of equations over $\mathbb{F}_2$. The operation

*For now you may think about α and β as being the same thing, and as you will soon see this is not far from the truth.

of the verifier on x and π is called *Verification*, for obvious reasons. We are now ready to delve deeper into these two sections of the proof, and we do so, beginning...

right now.

4.1 Arithmetization

First we give a few definitions that would be useful later on.

Definition 4.1. A boolean circuit ϕ of $\{and, not\}$ gates with t gates is a set of the boolean constraints of the following types:

- **AND gate:** $g_i = g_j \cdot g_k$ for some $j, k < i$.
- **NOT gate:** $g_i = g_j + 1$ for some $j < i$.

Where g_i is the output of gate i. If we also want the circuit to be satisfied we have the additional constraint :

- **Output gate:** $g_t = 1$

Where g_t is the output gate.

Definition 4.2. A boolean circuit ϕ is satisfiable if there exists an assignment A to the gates of ϕ that satisfies all the constraints in ϕ.

We can also represent these constraints as a set of equations over $\mathbb{F}_2$:

Definition 4.3. $\alpha \in \mathbb{F}_2^k$ satisfies ϕ if for every constraint ϕ_i in ϕ :

- ϕ_i *is an AND constraint:* $\alpha_i = \alpha_j \cdot \alpha_k$
- ϕ_i *is a NOT constraint:* $\alpha_i = \alpha_j + 1$
- *Output:* $\alpha_t = 1$

The last definition used quadric constraints to represent the AND gates of the circuit. Suppose, however, that the verifier is given an assignment α to ϕ, encoded into π via the Hadamard encoding for local testability. The Hadamard encoding, by definition, provides the verifier with an efficient way to query linear functions of α. Sadly, there is no efficient way for the verifier to query quadric functions of α, or any other function that is not linear for that matter (by efficient we mean that the verifier can tell the result of the function, using only a small, i.e. constant, number of queries). Thus we want to transform the AND constraints to a linear form, and we do so by converting the vector α to a matrix β, leading to the following definition :

Definition 4.4. $\beta \in \mathbb{F}_2^{k \times k}$ satisfies ϕ if the following holds :

- $\forall i,j : \beta_{i,j} = \beta_{i,i} \cdot \beta_{j,j}$
- ϕ_i *is an AND constraint:* $\beta_{i,i} = \beta_{j,k}$
- ϕ_i *is a NOT constraint:* $\beta_{i,i} = \beta_{j,j} + 1$
- *Output:* $\beta_{t,t} = 1$

From the definition it is easily seen that in fact, $\beta = \alpha \cdot \alpha^T$, which is known as the outer product $\alpha \otimes \alpha$. The constraints that are dependent on ϕ have become linear. The first constraint, which verifies that indeed $\beta = \alpha \cdot \alpha^T$, is independent of ϕ and can be verified efficiently. The following theorem tells us that Definition 4.3 and Definition 4.4 are equivalent.

Lemma 4.5. *$\alpha \in \mathbb{F}_2^k$ satisfies ϕ according to Definition 4.3 $\iff$ $\beta = \alpha \cdot \alpha^T$ and β satisfies ϕ according to Definition 4.4.*

Proof. For the first part, assume that $\alpha \in \mathbb{F}_2^k$ satisfies ϕ according to Definition 4.3, and let $\beta = \alpha \cdot \alpha^T$. We wish to show that β satisfies ϕ according to Definition 4.4. First, it is easy to see that since $\beta = \alpha \cdot \alpha^T$, we have $\alpha = diag(\beta)$, meaning $\forall i \beta_{i,i} = \alpha_i$. We also have $\forall i,j : \beta_{i,j} = \beta_{i,i} \cdot \beta_{j,j}$, thus the first constraint of Definition 4.4 is satisfied. Now, suppose that ϕ_i is an *AND* constraint, of the form $\beta_{i,i} = \beta_{j,k}$. Since α satisfies ϕ_i by Definition 4.3, we have $\beta_{i,i} = \alpha_i = \alpha_j \cdot \alpha_k = \beta_{j,k}$, so β satisfies ϕ_i. Suppose that ϕ_i is a *NOT* constraint, of the form $\beta_{i,i} = \beta_{j,j} + 1$. Again, since $\beta_{i,i} = \alpha_i$ and α satisfies ϕ_i, we have easily $\beta_{i,i} = \alpha_i = \alpha_j + 1 = \beta_{j,j} + 1$, and β satisfies this constraint as well. Finally, suppose that ϕ_i is the *Output* constraint, and again we have $\beta_{t,t} = \alpha_t = 1$.
For the second part, assume that $\beta = \alpha \cdot \alpha^T$ and β satisfies ϕ according to Definition 4.4. We wish to show that α satisfies ϕ according to Definition 4.3. For an *AND* constraint ϕ_i, we have as before $\alpha_i = \beta_{i,i} = \beta_{j,k} = \alpha_j \cdot \alpha_k$, so α satisfies this constraint. For a *NOT* constraint ϕ_i, we have $\alpha_i = \beta_{i,i} = \beta_{j,j} + 1 = \alpha_j + 1$, so α satisfies this constraint as well. For the *Output* constraint we have $\alpha_t = \beta_{t,t} = 1$, meaning it is also satisfied by α. Thus α satisfies all the constraints in ϕ. □

That completes our discussion over the encoding of the assignment to ϕ, and we are now ready for the next section of the proof.

4.2 Verification

So far we have seen how an assignment to ϕ is encoded into π. In this section we will show a PCP verifier which, given input x and oracle access to π, uses π to verify $x \in L$. To make things more compact, we give the following definition :

Definition 4.6. A linear oracle O is a linear function $O : \mathbb{F}_2^k \to \mathbb{F}_2$, meaning there exists $\beta \in \mathbb{F}_2^k$ such that for every $a \in \mathbb{F}_2^k$: $O(a) = \sum_{i=1}^{k} a_i \beta_i$

Note that since π is a Hadamard code word of β, π is also a linear oracle for β according to the previous definition, by viewing $\beta \in \mathbb{F}_2^{k \times k}$ as a long vector $\beta \in \mathbb{F}_2^{k^2}$. The PCP verifier is now described in the following main lemma :

Lemma 4.7. *(Arithmetization) There exists a PCP verifier V_{lin} such that for every circuit ϕ of size $t = f(n)$ and for every $\beta \in \mathbb{F}_2^{t^2}$, if V_{lin} has an oracle access to a linear oracle $O(\beta)$ of β and to a randomness source R, V_{lin} makes 4 queries to $O(\beta)$ and the following holds :*

- *Completeness : if β satisfies ϕ according to Definition 4.4 then*

$$Pr[V_{lin}^{O(\beta)}[\phi, R] = \mathsf{accept}] = 1$$

- *Soundness : if β does not satisfy ϕ according to Definition 4.4 then*

$$Pr[V_{lin}^{O(\beta)}[\phi, R] = \mathsf{reject}] \geq \tfrac{1}{4}$$

Proof. The verifier performs the following steps :

1. Randomly select $r, s \in \mathbb{F}_2^t$ uniformly and independently
2. Verify that $r \cdot \beta \cdot s = 0$ by making the following queries :
   - $Q_1 = \sum_{i=1}^{t} r_i \beta_{ii}$
   - $Q_2 = \sum_{j=1}^{t} s_j \beta_{jj}$
   - $Q_3 = \sum_{i,j=1}^{t} r_i s_j \beta_{ij}$

   and reject if $Q_1 \cdot Q_2 + Q_3 \neq 0$.
3. Randomly select a subset of the constraints $I \subseteq [t]$ uniformly and independently
4. Verify that the chosen constraints are satisfied by making the query :
   - $Q_4 = \sum_{i \in I} \phi_i(\beta)$

   and reject if $Q_4 \neq 0$. Otherwise accept.

From the above description it is clear that V_{lin} makes exactly 4 queries to $O(\beta)$. Thus the main arguments of the proof concern the completeness and (mostly) the soundness requirements.

Completeness Suppose that indeed $\beta = \alpha \cdot \alpha^T$ and α satisfies ϕ according to Definition 4.3. Thus we have

$$\forall r, s \in \mathbb{F}_2^t : (r \cdot \alpha) \cdot (\alpha^T \cdot s) = r \cdot (\alpha \cdot \alpha^T) \cdot s = r \cdot \beta \cdot s$$

Noting that Q_1 and Q_2 correspond to $r \cdot \alpha$ and $\alpha^T \cdot s$ respectively (remember that $\alpha_i = \beta_{ii}$), thus $Q_1 \cdot Q_2 = (r \cdot \alpha) \cdot (\alpha^T \cdot s)$, which by our assumption is equal to $r \cdot \beta \cdot s = Q_3$, and the test made at step 2 passes. Also, since all the constraints in ϕ are satisfied by β, then the sum of any subset of the constraints in ϕ is 0 given β and the test made at step 4 also passes, so V_{lin} accepts with a probability 1.

Soundness Suppose that β does not satisfy ϕ according to Definition 4.4. We divide the proof into cases :

1. $\beta \neq \alpha \cdot \alpha^T$, for $\alpha = diag(\beta)$.
2. $\beta = \alpha \cdot \alpha^T$, for $\alpha = diag(\beta)$, but α does not satisfy ϕ according to Definition 4.3.

For the first case, we define the difference matrix D to be $D = \alpha \cdot \alpha^T - \beta$, and by our assumption $D \neq 0$, that is, there exist indices (i, j) such that $D_{i,j} \neq 0$. We wish to show

$$\Pr_R[(r^T \cdot \alpha) \cdot (\alpha^T \cdot s) \neq r^T \cdot \beta \cdot s] \geq \tfrac{1}{4}$$

Note that $(r^T \cdot \alpha) \cdot (\alpha^T \cdot s) = r^T \cdot (\alpha \cdot \alpha^T) \cdot s$ and $r^T \cdot \beta \cdot s - r^T \cdot (\alpha \cdot \alpha^T) \cdot s = r^T \cdot D \cdot s$. We make the following claim :

Observation 4.8. For every matrix $D \neq 0$, $\Pr_{r,s}[r^T \cdot D \cdot s \neq 0] \geq \frac{1}{4}$

Proof. Since $\Pr_r[r^T \cdot D \neq 0] \geq \frac{1}{2}$, and in the same way, $\Pr_s[D \cdot s \neq 0] \geq \frac{1}{2}$, it follows that $\Pr_s[D \cdot s \neq 0 | r^T \cdot D \neq 0] \geq \frac{1}{4}$, meaning $\Pr_R[(r^T \cdot \alpha) \cdot (\alpha^T \cdot s) \neq r^T \cdot \beta \cdot s] \geq \frac{1}{4}$, which is what we wanted to show. □

For the second case, since not all the constraints of ϕ are satisfied, we know that $\phi_k(\beta) = 1$ for at least one index k. We will show that $\Pr[Q_4 = 1] = \Pr[\sum_{i \in I} \phi_i(\beta) = 1] \geq \frac{1}{2}$. First we define two sets $A, B \subseteq \mathcal{P}([t])$ by :
$\forall X \in \mathcal{P}([t])$:

$$X \in A \Leftrightarrow \sum_{i \in X} \phi_i(\beta) = 0$$
$$X \in B \Leftrightarrow \sum_{i \in X} \phi_i(\beta) = 1$$

And since I is chosen uniformly over $\mathcal{P}([t])$, showing $|A| \leq |B|$ will suffice. For that, consider the injective function $F : A \to B$, defined by :

$$F(X) = \begin{cases} X \setminus \{k\} & k \in X \\ X \cup \{k\} & k \notin X \end{cases}$$

First we need to show that the range of F is indeed B. Let $X \in A$ and thus $\sum_{i \in X} \phi_i(\beta) = 0$. If $k \in X$, we have $\sum_{i \in X} \phi_i(\beta) = 0 \Rightarrow \sum_{i \in X \setminus \{k\}} \phi_i(\beta) = 1 \Rightarrow \sum_{i \in F(X)} \phi_i(\beta) = 1$ and so $F(X) \in B$. If on the other hand $k \notin X$, we have $\sum_{i \in X} \phi_i(\beta) = 0 \Rightarrow \sum_{i \in X \cup \{k\}} \phi_i(\beta) = 1 \Rightarrow \sum_{i \in F(X)} \phi_i(\beta) = 1$, and so again $F(X) \in B$.
Now we show that F is injective. Let $X_1, X_2 \in A$, such that $X_1 \neq X_2$, and assume without loss of generality that for some j, $j \in X_1$ and $j \notin X_2$. Assume $j \neq k$, so $j \in F(X_1)$ and $j \notin F(X_2)$, thus $F(X_1) \neq F(X_2)$. Now assume $j = k$, so $j \notin F(X_1)$ and $j \in F(X_2)$, and again $F(X_1) \neq F(X_2)$, so F is injective.

To summarize, we have seen that in the first case $\Pr[V_{lin}^{O(\beta)}[\phi, R] = \mathsf{reject}] \geq \frac{1}{4}$, and in the second case $\Pr[V_{lin}^{O(\beta)}[\phi, R] = \mathsf{reject}] \geq \frac{1}{2}$. Assuming worst case yields $\Pr[V_{lin}^{O(\beta)}[\phi, R] = \mathsf{reject}] \geq \frac{1}{4}$, and our proof is complete. □

The formal proof of Theorem 3.1 is completed in Homework assignment 2.

4.3 Bibliographical notes

The notes are identical to those of our previous lecture. Theorem 3.1 was originally proved in Arora et al. [1998]. It uses the result stating that the Hadamard codes are locally testable, as proved in Blum et al. [1990].

Lecture 5

Composition of PCPs of Proximity (PCPP)

February 18th, 2008

Lecturer: Eli Ben-Sasson Scribe: Yevgeny Reznikov and Omer Karin

In order to progress from the exponential-length PCP described in the previous lectures, we will introduce the notion of *PCPs of Proximity* (PCPP) and present the theorem of *proof composition*. In the previous lectures, our non-adaptive PCP-verifier worked in the following manner: First, based on the random coins R it chose a group of queries I_R and built a decision circuit C_R. Then, it queried the proof-oracle for I_R, assigned input values to C_R (according to the oracle's responses) and decided whether to accept or reject according to the output of C_R. In *proof composition*, the verifier goes through the first two steps, but then, instead of querying the oracle, it recurses on C_R and I_R: The verifier now verifies that the random bits I_R are themselves a proof for C_R. For this the verifier will need oracle access to y, an implicit input that represents the input assignment to the original decision circuit (The one which the proof-oracle "proves" to be satisfying), and, in addition, the guarantee that the original verifier works under the conditions of *PCPP*. These conditions will now be presented.

5.1 PCPs of Proximity (PCPP)

Let L be a language in $\mathbf{NTIME}(f(n))$: There exists a non-deterministic Turing Machine M s.t. $L(M) = L$. We will use the following definitions:

Definition 5.1 (Pair-Language)**.** The language Pair-L is the language of all the pairs (φ, y) that are accepted by M (where φ is the input of M, and y is the "proof" of $\varphi \in L$).

$$\text{Pair-L} = \{(\varphi, y) : M(\varphi, y) = \mathsf{accept}\}$$

The language L_φ is the language of all valid proofs y for φ being in L.

$$L_\varphi = \{y : (\varphi, y) \in \text{Pair-L}\}$$

We denote by $\Delta(y, L_\varphi)$ how far y is from being a valid proof for φ being in L. It will be measured by the minimal distance of y from a valid proof y':

$$\Delta(y, L_\varphi) = \begin{cases} 1 & L_\varphi = \emptyset \\ \min_{y' \in L_\varphi} \Delta(y, y') & \text{otherwise} \end{cases}$$

where $\Delta(y, y')$ is the relative Hamming distance between y and y'.

The action of a PCP verifier V on a boolean circuit φ can be defined as a function from it's random bits R to the pair: $(I_R, \mathcal{C}_R)$, where I_R is a set containing the indexes which V has chosen to read from the oracles, and $\mathcal{C}_R$ is the decision circuit generated.

Definition 5.2 (PCPP Verifier)**.** A *PCPP Verifier* is a non-adaptive, randomized machine with access to two oracles, the first being an *input oracle*, and the second being a *proof oracle*. We denote by $V^{y,\pi}[\varphi, R]$ the output of the verifier on input φ, input oracle y, proof oracle π and randomness R. The output of the verifier is a pair: A set of indexes I_R (pointing to locations in y and π) and a decision circuit φ_R.

Definition 5.3 (PCPP Class)**.** Let PAIR-L be some pair language. Then, given a list of computational restrictions, a completeness function $c : \mathbb{N}^+ \to [0,1]$ and a soundness function $s : \mathbb{N}^+ \times [0,1] \to [0,1]$, it holds that:

$$\textsc{Pair-L} \in \mathbf{PCPP} \left(\text{computational restrictions} \;\middle|\; \begin{array}{l} c = 1 \\ s(n,\delta) \end{array} \right).$$

Iff there exists a PCPP verifier V which satisfies the computational restrictions, and for which, for all y, φ the following soundness and completeness requirements hold:

- **Completeness:** If $y \in L_\varphi$ there exists a proof π such that

$$\Pr_R[(y,\pi)|_{I_R} \in L_{\varphi_R}] = 1.$$

- **Soundness:** For every y and for every proof π,

$$\mathop{\mathrm{E}}_R[\Delta((y,\pi)|_{I_R}, L_{\varphi_R})] \geq s(n, \Delta(y, L_\varphi)).$$

Because the completeness is perfect, the soundness condition for $y \in L_\varphi$ simply requires that $s(n,0) = \mathop{\mathrm{E}}_R[0] = 0$.

We will now show that our PCP-verifier for SAT from Theorem 3.1 can be converted to a PCPP-verifier for the respective pair language PAIR-SAT.

Theorem 5.4.

$$\textsc{Pair-SAT} \in \mathbf{PCPP} \left(\begin{array}{lcl} q & = & O(1) \\ \Sigma & = & \mathbb{F}_2 \\ \ell(n) & = & 2^{n^2} \end{array} \;\middle|\; \begin{array}{lcl} c & = & 1 \\ s(n,\delta) & \geq & \frac{\delta}{10{,}000} \end{array} \right) hh.$$

Proof. The input of our PCPP verifier will be a boolean circuit φ, and it will have oracle access to the circuit's implicit input y and to a proof π of y satisfying φ. We will use the Hadamard-based PCP verifier from Theorem 3.1 (denoted V), so the proof π is defined accordingly. The PCPP verifier will first activate V to check if π is a valid proof for φ being satisfiable, i.e. for $\varphi \in$ SAT. If V rejects, then our new verifier will reject as well. If it

doesn't reject, then what is left to be checked is that π actually encodes y. To do so, a random bit from y is chosen and compared with the respective decoded bit from π (the bit will be locally decoded, using the method proved in Homework assignment 1). The verifier will accept iff the two bits are the same.
The computational restrictions hold because they hold for V, with some additional randomness and queries required for choosing and decoding the bits when comparing y and π. The completeness requirement holds since it holds for V, and since if π encodes y, then it's comparison to y will obviously succeed. For the soundness requirement, we will note the following: If L_φ is empty (ϕ isn't satisfiable) then π obviously cannot be an encoding of a satisfying assignment and when activating V the verifier will reject with probability greater than $\frac{1}{200}$. If $L_\varphi \neq \emptyset$, and the PCP verifier rejects with probability smaller than $\frac{1}{200}$, then π ought to be very close to encoding some satisfying assignment y'. But, since $\Delta(y, y') \geq \delta$, when choosing a random bit from y and comparing it to the respective bit in y', they will differ with probability at least δ. Thus, when activating the additional test comparing π and y, we will reject with probability close to δ.
The complete proof of this theorem is given as Homework assignment 3.

□

5.2 Proof Composition

We will next present the notion of *proof composition*. Recall that a PCPP verifier outputs a pair $(I_R, \mathcal{C}_R)$, and accepts iff the restriction of π to indices I_R, denoted $\pi|_{I_R}$, satisfies the decision circuit $\mathcal{C}_R$. The idea behind *proof composition* is to act recursively on $(I_R, \mathcal{C}_R)$, i.e., to prove that $\pi|_{I_R}$ satisfies $\mathcal{C}_R$ by using an "inner" PCPP verifier, thus reducing query complexity. We do the recursion on PCPPs and not on PCPs, because if we recurse on PCPs we get C_R to be a satisfiable circuit, so it will always have a satisfying assignment.

Theorem 5.5 (Proof Composition)**.** *For $s(n, \cdot)$ which is convex and monotonically nondecreasing for every n,*

$$\text{If } \textsc{Pair-SAT} \in \mathbf{PCPP} \left(\begin{array}{l|l} q(n) & \\ r(n) & \\ \ell(n) & c = 1 \\ t(n) & s(n, \delta) \\ d(n) & \end{array} \right)$$

$$\text{Then } \textsc{Pair-SAT} \in \mathbf{PCPP} \left(\begin{array}{l|l} q(d(n)) & \\ r(n) + r(d(n)) & \\ \ell(n) + 2^{r(n)} \cdot \ell(d(n)) & c = 1 \\ t(n) + t(d(n)) & s(d(n), s(n, \delta)) \\ d(d(n)) & \end{array} \right).$$

The theorem will be proved in the next lecture.

5.3 Bibliographical notes

The use of proof composition as a method for reducing query complexity was introduced in Arora and Safra [1998]. The original composition theorem was formulated in terms of PCPs and multi-prover interactive proofs (a concept we have not discussed in class). The notion of a PCPP was introduced in Ben-Sasson et al. [2004] and in Dinur and Reingold [2004] under the name "assignment testers" and used there to simplify the construction of PCP proofs and reduce their length. Similar notions to PCPPs appeared earlier in Ergün et al. [2000] and also in Szegedy [1999].

Lecture 6

Short PCPs based on PCPPs for Reed-Solomon codes

February 25th, 2008

Lecturer: Eli Ben-Sasson Scribe: Iddo Bentov

We progress towards a more efficient PCP theorem, in terms of the length of its proof oracle. We will achieve this result by using encodings that are shorter than Hadamard, and by composing PCPPs thereby obtaining PCPPs with a proof length of $n \cdot \text{polylog}(n)$. To date, all the PCP theorems with a subexponential proof length are obtained via composition. We start by proving the composition theorem.

6.1 Proof of the Composition Theorem

Recall Definition 5.3 and let us now prove:

Theorem 5.5 (restated). *For $s(n,\cdot)$ which is convex and monotonically nondecreasing for every n,*

$$\text{If } \textsc{Pair-SAT} \in \mathbf{PCPP} \left(\begin{array}{l|l} q(n) & \\ r(n) & \\ \ell(n) & c = 1 \\ t(n) & s(n,\delta) \\ d(n) & \end{array} \right)$$

$$\text{Then } \textsc{Pair-SAT} \in \mathbf{PCPP} \left(\begin{array}{l|l} q(d(n)) & \\ r(n) + r(d(n)) & \\ \ell(n) + 2^{r(n)} \cdot \ell(d(n)) & c = 1 \\ t(n) + t(d(n)) & s(d(n), s(n,\delta)) \\ d(d(n)) & \end{array} \right).$$

We note that $d(n) \triangleq \max_R |C_R|$ denotes the maximal size of a decision circuit C_R generated by $V^{y,\pi}[C,R]$ on input of size $|C| = n$ (C and C_R are φ and φ_R of Lecture 5). We expect $d(n)$ to be $o(n)$, for example $d(n) = \log(n)$ or $d(n) = \sqrt{n}$.

Proof. Define a new verifier $V^{y,\pi'}_{\text{composed}}[C, R \circ R']$ that expects a proof oracle of the concatenated form $\pi' = \pi \circ \{\pi_R : R \in \{0,1\}^{r(n)}\}$, where $\pi_R \leq \ell(d(n))$ holds for every R, and $|R'| = r(d(n))$.
Thus the length of the proof oracle of V_{composed} is indeed bounded by $\ell(n) + 2^{r(n)} \cdot \ell(d(n))$, and the randomness of V_{composed} is $r(n) + r(d(n))$, as required.
V_{composed} first invokes V on the input C by using the random coins R, and receives the output pair (C_R, I_R) that V generated. Note that $|C_R| \leq d(n)$. Now V_{composed} invokes

$V^{y_R,\pi_R}[C_R, R']$, i.e. it runs V with C_R as its input, with the random coins R', and with $y_R \triangleq (y,\pi)|_{I_R}$ and π_R as its input and proof oracles. The output of V_{composed} shall be the output that V generates in this second invocation, and thus the total running time of V_{composed} is bounded by $t(n) + t(d(n))$, and the size of the decision circuit that V_{composed} generates for its output is bounded by $d(d(n))$. Because V is nonadaptive, V_{composed} can generate the set of indices I_R without actually querying the oracles during the first invocation of V, and therefore the total number of queries that V_{composed} makes is the number of queries that V makes in the second invocation, which is bounded by $q(d(n))$. Thus all the required computational restrictions hold.

Completeness: If $y \in L_C$ then there exists a proof π such that for every R it holds that C_R is satisfied by $y_R = (y,\pi)|_{I_R}$. Therefore, by our assumption on Pair-SAT, there exists a proof π_R, $|\pi_R| \leq \ell(d(n))$, such that $V^{y_R,\pi_R}[C_R, R']$ accepts for every R'. So for the concatenated proof $\pi' = \pi \circ \{\pi_R : R \in \{0,1\}^{r(n)}\}$, where each π_R is the proof that is guaranteed to exist for C_R, it holds that $V^{y,\pi'}_{\text{composed}}[C, R \circ R']$ always accepts.

Soundness: Suppose $\Delta(y, L_C) = \delta > 0$, so for every proof π we have $\mathop{\mathrm{E}}_{R}[\Delta(y_R, L_C)] \geq s(n,\delta)$, and for any concatenation of small proofs $\{\pi_R : R \in \{0,1\}^{r(n)}\}$ we have

$$\mathop{\mathrm{E}}_{R,R'}[\Delta((y_R,\pi_R)|_{I_{R'}}, L_{C_{R'}})] \overbrace{=}^{\text{linearity of expectation}} \mathop{\mathrm{E}}_{R}[\mathop{\mathrm{E}}_{R'}[\Delta((y_R,\pi_R)|_{I_{R'}}, L_{C_{R'}})]].$$

Let $\delta_R \triangleq \Delta(y_R, L_{C_R})$. For a fixed R, it holds that $\mathop{\mathrm{E}}_{R'}[\Delta((y_R,\pi_R)|_{I_{R'}}, L_{C_{R'}})] \geq s(d(n), \delta_R)$ according to Definition 5.3, and therefore,

$$\mathop{\mathrm{E}}_{R,R'}[\Delta((y_R,\pi_R)|_{I_{R'}}, L_{C_{R'}})] \geq \mathop{\mathrm{E}}_{R}[s(d(n),\delta_R)] = \sum_R \Pr(R) s(d(n),\delta_R) \overbrace{\geq}^{\text{convexity}} s(d(n), \sum_R \Pr(R)\delta_R)$$

$$= s(d(n), \mathop{\mathrm{E}}_{R}[\delta_R]) = s(d(n), \mathop{\mathrm{E}}_{R}[\Delta(y_R, L_C)]) \overbrace{\geq}^{\text{monotonicity}} s(d(n), s(n,\delta))$$

□

6.2 PCPPs for Reed-Solomon Codes

Our next objective is to prove, using PCPPs, the following theorem:

Theorem 6.1 (Quasilinear PCP)**.**

$$\text{SAT} \in \mathbf{PCP}\left(\begin{array}{lcl|lcl} \ell & = & n \cdot \text{polylog}(n) & & & \\ q & = & O(1) & & & \\ \Sigma & = & \mathbb{F}_2 & c & = & 1 \\ \textsf{nonadaptive} & & & s(n) & \geq & \frac{1}{\text{polylog}(n)} \\ t & = & n^{O(1)} & & & \\ r & = & \log(\ell) + O(1) & & & \end{array}\right).$$

Note: $\ell = n \cdot \text{polylog}(n)$ is beyond the scope of this course. Instead, we will settle for $\ell = n^2 \cdot \text{polylog}(n)$.

To obtain PCPs with short proof lengths, as in Theorem 6.1, we obviously cannot use the exponential size Hadamard encoding. Therefore, we now wish to encode proofs with a more efficient code than Hadamard. We will use Reed-Solomon codes, which do not have good local testability properties in comparison to the Hadamard code. However, with the use of PCPPs we shall achieve the same effect as that of a local tester. Namely, by using a few queries we will be able to distinguish with high probability between codewords and words that are far (in Hamming distance) from the code.

Definition 6.2 (Reed-Solomon code)**.** Let $\mathbb{F}$ be a finite field, $|\mathbb{F}| = n$. The Reed-Solomon code $\text{RS}(k, \mathbb{F})$ transforms a message $(a_0, a_1, \ldots, a_{k-1}) \in \mathbb{F}^k$ to the codeword $\left\langle \sum_{i=0}^{k-1} a_i x^i | x \in \mathbb{F} \right\rangle \in \mathbb{F}^n$.

Thus the message length is k, the blocklength is n, the alphabet is $\mathbb{F}$, and the code distance is $n - k + 1$ because two distinct polynomials of degree $k - 1$ can agree on at most $k - 1$ points, so the rest of the coordinates must differ. Reed-Solomon is a linear code, because $\sum_{i=0}^{k-1} a_i x^i + \sum_{i=0}^{k-1} b_i x^i = \sum_{i=0}^{k-1} (a_i + b_i) x^i$. Reed-Solomon is also a locally testable code, in the sense that there is exactly one polynomial of degree $k - 1$ that passes through k points, so $k + 1$ queries can be used to test a code word. However, in our setting k will be too large a query complexity to tolerate, so we shall rely on PCPPs for the natural family of pairs induced by the Reed-Solomon code.

Definition 6.3 (Pair-RS)**.** PAIR-RS $= \{((\mathbb{F}, k), p : \mathbb{F} \to \mathbb{F})\}$ where for every such $((\mathbb{F}, k), p)$:

- $k < \frac{|\mathbb{F}|}{10}$
- $\deg(p) < k$ (in other words, $p \in \text{RS}(\mathbb{F}, k)$)

For a pair $((\mathbb{F}, k), p : \mathbb{F} \to \mathbb{F})$, we regard its first element $(\mathbb{F}, k)$ as the explicit input of the verifier, and its second element $p : \mathbb{F} \to \mathbb{F}$ as the implicit (oracle) input of the verifier.

Definition 6.4 (Pair-bin-RS)**.** PAIR-BIN-RS $\subsetneq$ PAIR-RS is the language for which we require $\mathrm{char}(\mathbb{F}) = 2$, i.e. $|\mathbb{F}| = 2^q$ for some $q \in \mathbb{N}$, and we can regard the elements of $\mathbb{F}$ as q-tuples of bits.

Theorem 6.5.

$$\text{PAIR-BIN-RS} \in \mathbf{PCPP} \left(\begin{array}{lcl|lcl} \ell & = & n \cdot \mathrm{polylog}(n) & & & \\ q & = & O(1) & c & = & 1 \\ \Sigma & = & \mathbb{F} & s(n,\delta) & \geq & \frac{\delta}{\mathrm{polylog}(n)} \\ \mathsf{nonadaptive} & & & & & \\ t & = & n^{O(1)} & & & \end{array} \right)$$

Where $n = |\mathbb{F}|$ (or $|\mathbb{F}| \log |\mathbb{F}|$ in bits).

The verifier will use this PCPP as a replacement for the local testability properties of the Hadamard code. Namely, the verifier will use this PCPP to test whether certain polynomials that are encoded in the proof are Reed-Solomon codewords, i.e. whether these polynomials meet the required condition on their maximal degree.
Note: the proof of this theorem is beyond our scope, so we will use it as a black box instead.

6.3 PCPPs for Vanishing Reed-Solomon codes

Let us now define an additional PCPP, which will be a crucial building block in the proof of Theorem 6.1. Specifically, this PCPP will be used to reduce the problem of checking generic SAT constraints to the problem of checking whether a single polynomial vanishes on many points.

Definition 6.6 (Pair-Vanishing-RS)**.** PAIR-VANISHING-RS $= \{((\mathbb{F}, k, H), p : \mathbb{F} \to \mathbb{F})\}$ where for every such $((\mathbb{F}, k, H), p)$:

- $H \subseteq F$
- $((\mathbb{F}, k), p) \in$ PAIR-RS
- $\forall \alpha \in H : p(\alpha) = 0$

Theorem 6.7.

$$\text{If } \text{PAIR-BIN-RS} \in \mathbf{PCPP} \left(\begin{array}{lcl|lcl} \ell_{RS} & = & n \cdot \mathrm{polylog}(n) & & & \\ q_{RS} & = & O(1) & c & = & 1 \\ \Sigma & = & \mathbb{F} & s_{RS}(n,\delta) & \geq & \frac{\delta}{\mathrm{polylog}(n)} \\ \mathsf{nonadaptive} & & & & & \\ t_{RS} & = & n^{O(1)} & & & \end{array} \right)$$

Then PAIR-VANISHING-RS $\in$ **PCPP** $\left(\begin{array}{lcl|lcl} \ell_{VRS} & = & n + 2 \cdot \ell_{RS} & & & \\ q_{VRS} & = & 2 \cdot q_{RS} + O(1) & c & = & 1 \\ \Sigma & = & \mathbb{F} & s_{VRS}(n, \delta) & \geq & \frac{s_{RS}(n,\delta)}{10} \\ t_{VRS} & = & O(t_{RS}) & & & \end{array} \right)$.

This theorem will be proven in the next lecture.

4 Bibliographical Notes

Bibliographical notes regarding proof composition appeared in our previous lecture, see Section 5.3. The short PCPPs for Reed-Solomon and vanishing Reed-Solomon codes stated in Theorem 6.5 and Theorem 6.7 respectively, are from Ben-Sasson and Sudan [2005].

Lecture 7

Short PCPs based on PCPPs for Reed-Solomon codes — Vanishing RS-codes and Arithmetization

March 3rd, 2008

Lecturer: Eli Ben-Sasson Scribe: Omer Karin

In this lecture we will finish the proof of Theorem 6.1. Using the black box PCPP verifier for Pair-Binary-RS, we will construct a PCPP verfier for the language Pair-Binary-VRS. With those verifiers at hand, we will be able to build a PCP verifier with short proof-length for a (complete) algebraic language ACSP, thus proving the theorem.

7.1 Pair-Binary-VRS

In the last lecture we defined the code $\mathrm{RS}(\mathbb{F}, k)$, as the language of all functions $P : \mathbb{F} \to \mathbb{F}$ such that P represents the evaluation of some polynomial of degree less than k on $\mathbb{F}$. The language Pair-Binary-RS was defined as the language of all triplets $(\mathbb{F}, k, P : \mathbb{F} \to \mathbb{F})$ such that $\mathrm{char}(\mathbb{F}) = 2$, $k \leq \frac{|\mathbb{F}|}{10}$ and $P \in RS(\mathbb{F}, k)$. We have also stated Theorem 6.5 that says Pair-Binary-RS has PCPPs with quasilinear length. We also defined (in Definition 6.6) the language Pair-Binary-VRS that is very similar to Pair-Binary-RS, but with the additional requirement that the polynomial p vanishes on all points of a set H (the set H is also supplied in the input). Our first step in this lecture is to prove that Pair-Binary-VRS also has short PCPPs. We restate the relevant theorem from Lecture 6.

Theorem 6.7 (restated).

$$\text{If } \textsc{Pair-bin-RS} \in \mathbf{PCPP} \left(\begin{array}{lcl|lcl} \ell_{RS} & = & n \cdot \mathrm{polylog}(n) & c & = & 1 \\ q_{RS} & = & O(1) & s_{RS}(n,\delta) & \geq & \frac{\delta}{\mathrm{polylog}(n)} \\ \Sigma & = & \mathbb{F} & & & \\ \textsf{nonadaptive} & & & & & \\ t_{RS} & = & n^{O(1)} & & & \end{array} \right)$$

$$\text{Then } \textsc{Pair-Vanishing-RS} \in \mathbf{PCPP} \left(\begin{array}{lcl|lcl} \ell_{VRS} & = & n + 2 \cdot \ell_{RS} & c & = & 1 \\ q_{VRS} & = & 2 \cdot q_{RS} + O(1) & s_{VRS}(n,\delta) & \geq & \frac{s_{RS}(n,\delta)}{10} \\ \Sigma & = & \mathbb{F} & & & \\ t_{VRS} & = & O(t_{RS}) & & & \end{array} \right).$$

Proof. From basic algebra, it holds that for some $\alpha \in \mathbb{F}$ and some polynomial $P : \mathbb{F} \to \mathbb{F}$, α

is a root of P iff $(x-\alpha) \mid P(x)$, i.e iff:

$$\exists \widetilde{P}(x), \quad \deg \widetilde{P} = \deg P - 1, \quad \text{s.t} \quad \widetilde{P}(x)(x-\alpha) = P(x)$$

We define $P_H(x) \triangleq \prod_{h \in H}(x-h)$. From the above it follows that P vanishes on H iff:

$$\exists \widetilde{P}(x), \quad \deg \widetilde{P} = \deg P - |H|, \quad \text{s.t} \quad \widetilde{P}(x)P_H(x) = P(x)$$

We will now begin proving the theorem, assuming we are given the PCPP verifier V_{RS} for Pair-Binary-RS.
The PCPP verifier V_{VRS} for Pair-Binary-Vanishing-RS will be defined as following: Let π_{RS} be the proof (for the verifier V_{RS}) for $((\mathbb{F}, k), P) \in$ Pair-Binary-RS, let $\widetilde{P}$ be the polynomial for which $\widetilde{P}(x)P_H(x) = P(x)$ and let $\widetilde{\pi}_{RS}$ be the proof for $((\mathbb{F}, k-|H|), \widetilde{P}) \in$ Pair-Binary-RS. The proof for $((\mathbb{F}, k, H), P) \in$ Pair-Binary-VRS will be the concatenation of $\widetilde{\pi}_{RS}, \pi_{RS}, \widetilde{P}$.
V_{VRS} will activate $V_{RS}^{P,\pi}[\mathbb{F}, k]$ and $V_{RS}^{\widetilde{P},\widetilde{\pi}}[\mathbb{F}, k]$, and will reject if either of the two rejects. If they both accept, it will uniformly pick a field element $\alpha \in \mathbb{F}$, and will reject if $\widetilde{p}(\alpha)p_H(\alpha) \neq p(\alpha)$. If all three tests ended successfully, it will accept. Completeness holds from the discussion in the beginning (and because the verifier V_{RS} works with perfect completeness), and the computational requirements are obviously held.
Soundess: We assume that P is δ-far from any polynomial of degree less than k which vanishes on H. There are two options:

1. If P is $\frac{\delta}{100}$ far from any polynomial of degree less than k, then $V_{RS}^{P,\pi}[\mathbb{F}, k]$ will reject with probability greater than $\frac{\delta}{\text{polylog}(n)}$.

2. P is $\frac{\delta}{100}$-close to some polynomial of degree less than k, this polynomial, denoted $\hat{P}$, must be unique because $k < |\mathbb{F}|/10$. Furthermore, by assumption $\hat{P}$ does not vanish on H. There are two final subcases to consider.

   (a) If $\widetilde{P}$ is $\frac{1}{100}$-far from all polynomials of degree $k-|H|$, then $V_{RS}^{\widetilde{P},\widetilde{\pi}}[\mathbb{F}, k]$ will reject with probability high enough.

   (b) Otherwise, $\widetilde{P}$ is $\frac{1}{100}$-close to a unique polynomial of degree $k-|H|$, so $\widetilde{P}P_H$ is $\frac{1}{100}$ close to a polynomial of degree k, $\hat{\hat{P}}$. Since $\hat{\hat{P}}$ vanishes on H and $\hat{P}$ doesn't vanish on H, it must be that $\hat{P} \neq \hat{\hat{P}}$, and they disagree on at least $\frac{9}{10}$ of the points in $\mathbb{F}$ (because $k < \frac{|\mathbb{F}|}{10}$). Thus, in this very last case the rejection probability of the third test of V_{VRS} is at least:

$$1 - \frac{1}{10} - 2 \cdot \frac{1}{100} > \frac{1}{2} > \frac{\delta}{\text{polylog}(n)}$$

This completes the soundness analysis and with it Theorem 6.7 is proved. □

7.2 Algebraic Constraint SAT problem (ACSP)

We the PCPPs for PAIR-BINARY-RS and PAIR-BINARY-VRS we have the analog of the local tester and decoder for Hadamard codes used to construct exponential length PCPs. The final part is an arithmetic **NP**-complete language that "goes along" with RS-codes. This language is similar to the language MATRIXSAT defined in Homework assignment 2 and used in PCPs that are based on Hadamard codes.

Definition 7.1 (ALGEBRAIC CONSTRAINT SATISFACTION PROBLEM (ACSP))**.** An *instance* of size t of ACSP is a tuple $\phi = (\mathbb{F}, H, L_1, L_2, L_3, C^0, C^1)$ where

- $\mathbb{F}$ is a finite field of characteristic 2 and $100dt^2 < |\mathbb{F}| \leq 200dt^2$, where d is an absolute constant (defined in Theorem 2).
- $H \subset \mathbb{F}, |H| = t^2$.
- L_1, L_2, L_3 are linear functions from $\mathbb{F}$ to $\mathbb{F}$, i.e. $L_i : \mathbb{F} \to \mathbb{F}$ is given by $L_i(x) = a_i x + b_i$.
- C^0 is a univariate polynomial of degree at most d.
- C^1 is a polynomial in four variables, $C^1(x, y_1, y_2, y_3)$ where $\deg_x(C^1) \leq |H|$ and $\deg_{y_i}(C^1) \leq d$.

An *assignment* to ϕ is a univariate polynomial $A, \deg(A) \leq |H|$. We say that A satisfies ϕ iff

- $C^0(A(h)) = 0$ for all $h \in H$.
- $C^1(h, A(L_1(h)), A(L_2(h)), A(L_3(h)) = 0$ for all $h \in H$.

The language ACSP contains all satisfiable instances.

We show the **NTIME** (t)-completeness of ACSP by reducing the **NTIME** (t)-complete language DOMINO to it. The reduction and completion of the proof of Theorem 6.1 is given as Homework assignment 4.

3 Bibliographical Notes

The quasilinear PCPPs based on RS- and VRS-codes presented in this and the previous lecture appeared in Ben-Sasson and Sudan [2005].

Lecture 8

PCP Proofs Using Gap Amplification

March 10th, 2008

Lecturer: Eli Ben-Sasson Scribe: Gitit Raz and Amit Feller

8.1 Introduction

We saw the following:

Theorem 8.1.

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} \ell(n) & = & f(n)\text{polylog}(f)(n) & & & \\ q & = & O(1) & c & = & 1 \\ \Sigma & = & \mathbb{F}_2 & s & \geq & \frac{1}{\text{polylog}(()f(n))} \\ t(n) & = & f(n)^O(1) & & & \end{array}\right).$$

The notation for the list of restrictions is the same as in Theorem 1.3.
Actually: q=2, /Σ/=O(1)

Our next topic is a recent new proof of the PCP Theorem due to Dinur [2007]. The key step of this proof, the *Gap Amplification Theorem* stated next, is a reduction that retains query complexity and alphabet size but doubles the soundness. Using this reduction, we can start with a simple PCP that has small alphabet size and query complexity, but very weak soundness. Applying the gap amplification lemma enough times, we obtain the PCP Theorem. Additionally, if we start with a relatively sound PCP system as in Theorem 8.1 then we can obtain PCPs with constant soundness and query complexity and quasilinear length proofs.

8.2 Gap Amplification

In this lecture (and the next) we will focus on proving the following theorem. It says that if a language L has a PCP verifier with a certain setting of parameters, and the soundness obtained by this verifier is not so good, then we can create a new PCP verifier, with almost no changes to parameters, and with double the soundness. Informally, we double soundness while slightly decreasing other parameters (such as proof-length).

Theorem 8.2 (Gap Amplification Theorem)**.** *There exists a constant size alphabet Σ and*

constant $s_{\max} > 0$ *such that*

$$\mathbf{PCP}\left(\begin{array}{l|l} \ell = \ell_0(n) & \\ q = 2 & \\ \Sigma & c = 1 \\ r = r_0(n) & s = s_0(n) \\ \text{nonadaptive} & \end{array}\right) \subseteq \mathbf{PCP}\left(\begin{array}{l|l} \ell_1(n) = O(\ell_0(n)) & \\ q = 2 & \\ \Sigma & c = 1 \\ r_1(n) = r_0(n) + O(1) & s(n) \geq \min\{2s_0(n), s_{\max}\} \\ \text{nonadaptive} & \end{array}\right).$$

A few comments regarding the previous theorem are due.

- Notice soundness doubles on the right side of the containment.
- The alphabet size can probably be taken to be 3.

First, we notivce Theorem 8.2 implies PCPs with constant soundness, query complexity and quasilinear length proofs.

Corollary 8.3.

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} \ell(n) & = & f(n)\text{polylog}(f(n)) & c & = & 1 \\ q & = & 2 & s & > & 0 \\ |\Sigma| & = & O(1) & & & \end{array}\right).$$

The notation for the list of restrictions is the same as in Theorem 1.3.

Proof. Start from Theorem 8.1 and apply Theorem 8.2 $O(\log \log n)$ times so that the length of the final proof will increase by a factor of $2^{O(\log \log n)} = \text{polylog}(n)$ and the soundness will increase by a factor of $2^{O(\log \log n)} = \text{polylog}(n)$. □

Next we notice that Theorem 8.2 implies a query-efficient, constant-soundness PCP Theorem, albeit with polynomial length proofs.

Corollary 8.4.

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} \ell(n) & = & f(n)^{O(1)} & c & = & 1 \\ q & = & 2 & s > 0 & & \\ |\Sigma| & = & O(1) & & & \end{array}\right).$$

The notation for the list of restrictions is the same as in Theorem 1.3.

Proof. Start with an $\mathbf{NTIME}\left(f(n)\right)$-complete language L that is defined over constraints over two-variables, like 3COLOR. Since every $x \notin L$ has the property that every assignment

falsifies at least one constraint, we get

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} \ell(n) & = & f(n) & c & = & 1 \\ q & = & 2 & s(n) & \geq & \frac{1}{f(n)} \\ |\Sigma| & = & O(1) & & & \end{array}\right).$$

Use Theorem 8.2 $\log(f(n))$ times. We will get the soundness of $s_{\max}$ and a maximal length of $\ell_0(n) \geq f(n)^{O(1)}$ □

To summarize, we have seen two corollaries from Theorem 8.2:

1. We got the PCP theorem easily by using Theorem 8.2 $\log(n)$ times.
2. We got a PCP verifier with very short proofs.

Intuitively, what we have is soundness amplification in return for a longer proof. If we start off from a good starting point (good soundness, short proofs - which is Theorem 8.1), then we only need to use Theorem 8.2 a small number of times to get to Corollary 8.3.

8.3 Proving The Gap Amplification Theorem

We are stepping towards proving Theorem 8.2. The main part of the proof is the gap amplification. To prove this theorem it will be useful to argue about *constraint graphs* defined next.

Definition 8.5 (Constraint-Graph)**.** A Constraint-Graph (CG) is a triple $\mathcal{G} = (G, \Sigma, \mathcal{C})$ where $G = (V, E)$ is an undirected graph, possibly containing self loops and multiple edges. Σ is an alphabet. $\mathcal{C}$ is a group of constraints, one for each edge: $\mathcal{C} = \{C_e : \Sigma \times \Sigma \to \{\mathsf{accept}, \mathsf{reject}\} | e \in E\}$

Definition 8.6 (Assignment)**.** An assignment is a function $A : V \to \Sigma$, where each vertex is assigned a letter from the alphabet.

Definition 8.7 (Soundness of an Assignment)**.** We will define the soundness of an assignment A, as the fraction of the constraints that are not satisfied:

$$s(A, \mathcal{G}) = \Pr_{e \in E, e(u,v)}[C_e(A(u), A(v)) = \mathsf{reject}].$$

The soundness of a constraint graph is the minimal soundness of some assignment for it,

$$s(\mathcal{G}) = \min_{A:V\to\Sigma} S(A, \mathcal{G}).$$

For our next definition we need to recall the notion of a *promise problem*. We can look at a language as a partition of the set of words Σ^* into YES-instances (belonging to the language), and NO-instance (that don't belong to the language). A promise problem is also a partition of Σ^*, but into three sets, the third set being DON'T CARE-instances. The

intuition behind a promise problem is that it is a relaxation of a decision problem in which our algorithm needs to succeed only on the words we care about.

Definition 8.8 (Gap-CG)**.** Let n be the number of edges or constraints in a constraint graph $\mathcal{G}$. For a soundness function $\hat{s} : \mathbb{N}^+ \to [0, 1]$ the language Gap-CG (Σ,S) is a promise problem defined by

- YES: $\{\mathcal{G} = (G, \Sigma, C) | s(G) = 0\}$.
- NO: $\{\mathcal{G} = (G, \Sigma, C) | s(G) \geq \hat{s}(n)\}$.

Theorem 8.9 (Theorem 8.2 restated with constraint graphs)**.** *For every large enough Σ, there exists a constant $s_{\max} = s_{\max}(|\Sigma|) > 0$ such that:* Gap-CG (Σ,$s(n)$) *is reducible to* Gap-CG $(\Sigma, \min(2s(n), s_{\max}(|\Sigma|)))$*, in polynomial time, and the reduction increases graph size only by a constant.*

Proof. We prove that there exist constants $c_1, c_2, c_3 > 1$ such that $\frac{c_2}{c_1 \cdot c_3} \geq 2$ and the following three reductions hold. An arrow designates a polynomial time reduction that blows up the size of the constraint graph by a constant. We follow the diagram with an overview of the three steps.

$$\begin{gathered}
\text{Gap-CG}\left(\Sigma, s_0(n) = s(n)\right) \\
\downarrow \\
\text{Expander-Gap-CG}_{d,\frac{1}{2}}\left(\Sigma, s_1(n) \geq \frac{s_0(n)}{c_1}\right) \\
\downarrow \\
\text{Gap-CG}\left(\hat{\Sigma}(|\hat{\Sigma}| >> |\Sigma|), s_2(n) \geq c_2 \cdot s_1(n)\right) \\
\downarrow \\
\text{Gap-CG}\left(\Sigma, s_3(n) \geq \frac{s_2(n)}{c_3}\right)
\end{gathered}$$

Expanding In this first stage we rearrange the graph and make it an *expander graph* (defined below), at the cost of decreasing the soundness by a factor of c_1. This restructuring step is a combinatorial analog of the arithmetization process described in previous lectures.

Amplification In the second and main stage, we increase the soundness by c_2, but the alphabet will also increase by a constant factor.

Alphabet reduction In this last stage we decrease alphabet size by composition with a small-alphabet PCPP, but once again the soundness decreases by a factor c_3.
Summing up over the three steps, the graph size increases by a constant, the alphabet stays the same, and the soundness increases by $\frac{c_2}{c_1 \cdot c_3} \geq 2$. □

8.4 First step — Reduction to expander constraint graphs

Expander graphs have numerous applications in computer science and in mathematics. The construction of these graphs and the study of their properties are receiving a lot of attention in recent years. In what follows, we define these graphs only in terms of the combinatorial properties that will be needed in our proof.

Definition 8.10 (Expander Graph)**.** A graph G is a said to be a (d, λ)-expander if G is d-regular (each vertex has exactly d edges connected to it) and the following conditions hold

1. Vertex expansion - $\forall S \subseteq V, |S| \leq \frac{|V|}{2} : |E(S, \overline{S})| \geq \frac{d-\lambda}{2} * |S|$
2. Random walk expansion - $\forall F \subseteq E, \forall i \geq 0$: the probability that a random walk starting from a random edge in F will do the $i+1$ step inside F is at most $\frac{\lambda}{d}^i + \frac{|F|}{|E|}$

Two brief bits of information about the expansion parameter λ in the previous definition.

- It is known that λ is at least $2\sqrt{d-1}$. Explicit constructions of graphs achieving this λ are known, and these graphs are known as *Ramanujan expanders*.
- The smaller λ is in relation to d, the better.

We shall need to use expander graphs in our construction. The following Theorem states that such graphs can be constructed. The proof of this theorem can be found, e.g., in Margulis [1973]; Gabber and Galil [1981]; Lubotzky et al. [1988]; Reingold et al. [2000].

Theorem 8.11. *For each $\epsilon > 0$, there exists d such that for every n, it is possible to create a graph that is $(d, \epsilon * d)$-expander with size n, and in* poly(n) *time.*

Definition 8.12 ($Gap - CG_{(d,\lambda)}(\Sigma, s(n))$)**.** $Gap - CG_{(d,\lambda)}(\Sigma, s(n))$ is the promise problem that includes all the graphs that are (d, λ)-expanders.

The very first step in our proof of Theorem 8.9 is given by the following lemma. Its proof is part of Homework assignment 5.

Lemma 8.13. *There exists $c_1, d >$ such that there is a polynomial time and linear size reduction for: $Gap - CG(\Sigma, s_0(n)) \rightarrow Gap - CG_{(d,\frac{1}{2})}(\Sigma, s_1(n) \geq \frac{s_0(n)}{c_1})$*

8.5 Bibliographical Notes

The proof of the PCP Theorem by Gap Amplification, which is the topic of this and our next lecture, appeared in Dinur [2007]. An exposition of this proof can be found in Radhakrishnan and Sudan [2007].

LECTURE 9

GAP AMPLIFICATION II — SOUNDNESS AMPLIFICATION

MARCH 17th, 2008

LECTURER: Eli Ben-Sasson SCRIBE: Viaceslav Chernoy

We start by recalling the main theorem.

Theorem 9.1 (main)**.** *For every sufficiently big* Σ*, there is a constant* $s_{max} = s_{max}(\Sigma) > 0$*, such that* GAP-CG $(\Sigma, s(n))$ *problem may be polynomially reduced to* GAP-CG $(\Sigma, \min(s_{max}, 2s(n))$ *problem.*

Note. For small Σ, say $|\Sigma| = 2$, the theorem is not true, unless $\mathbf{P} = \mathbf{NP}$.

The proof of the theorem consists of three polynomial time reductions depending on constants $c_1, c_2, c_3 > 1$:

$$\begin{array}{c} \text{GAP-CG}\,(\Sigma, s_0(n)) \\ \downarrow \\ \text{EXPANDER-GAP-CG}_{d,\frac{d}{2}}\left(\Sigma, s_1(n) \geq \frac{s_0}{c_1}\right) \\ \downarrow \\ \text{GAP-CG}\left(\hat{\Sigma}, s_2(n) \geq \min\{s_{max}, c_2 \cdot s_1(n)\}\right) \\ \downarrow \\ \text{GAP-CG}\left(\Sigma, s_3(n) \geq \frac{s_2(n)}{c_3}\right) \end{array}$$

Arguing that $\frac{c_2}{c_1 \cdot c_3} \geq 2$ completes the proof of the theorem.

In the previous lecture and in Homework assignment 5 we showed the first reduction that builds a d-regular graph from any constraint graph. The main idea was to extend the graph by adding $O(1)$ edges with weight 0. In this case the soundness decreases by a factor of c_1. In this lecture we discuss the second reduction, which is presented in Section 9.1. Its validity is proved sketchily in Section 9.2.

9.1 The reduction

Assume we are given a d-regular constraint graph $\mathcal{G} = (G(V, E), \Sigma, C)$ with loops and an even constant $t > 0$. The reduction builds a constraint graph $\mathcal{G}^t$ that depends on an integer parameter t.

Definition 9.2 (Constraint Graph $\mathcal{G}^t$)**.** Constraint graph $\mathcal{G}^t$ is a triple $(G^t(V, E^t), \Sigma^{d^t}, C^t)$, where E^t and C^t are defined as follows. For every path $p = v_0 \to v_1 \to \ldots \to v_t$ of length t in the graph G, multi-set E^t contains an edge $\bar{e} = (v_0, v_t)$, and set C^t contains a constraint

$C_{\bar{e}} : \Sigma^{d^t} \times \Sigma^{d^t} \to \{\mathsf{accept}, \mathsf{reject}\}$. Where $C_{\bar{e}}(l_1, l_2) = \mathsf{accept}$ iff the assignment of l_1, l_2 to balls of radius t with centers in v_0, v_t is consistent with the path p and satisfies all constraints $C_{(v_i,v_{i+1})}$.

We say that the edge $\bar{e}$ *is created by the path* p. Sometimes we make no distinction between the edge in G^t and the path in G.

Definition 9.3 (Assignment A^t for $\mathcal{G}^t$)**.** An assignment for a constraint graph $\mathcal{G}^t$ is a function $A^t : V \to \Sigma^{d^t}$.

Consider a vertex $v \in V$. All vertices reached from v by some path of the length r form a ball of radius r. Formally, $\mathrm{Ball}_v(r) = \{u \in V | \exists p = v \to \ldots \to u \in G, |p| = r\}$. Clearly, $|\mathrm{Ball}_v(r)| = \frac{d}{d-2}((d-1)^r - 1) \leq d^r$.
Informally, $A^t(v) = (\sigma_1, \ldots, \sigma_{d^t}) \in \Sigma^{d^t}$ means that vertex v *has opinion* regarding values assigned by A^t to all the vertices in $\mathrm{Ball}_v(t)$. Let's denote by $A^t_u(v) = \sigma$ the opinion of v regarding a value assigned to u. Then A^t satisfies $C_{\bar{e}}$, iff the opinions of v_0 and v_t regarding values assigned to vertices v_i in the path are consistent, i.e., $\forall 0 \leq i \leq t, A^t_{v_i}(v_0) = A^t_{v_i}(v_t)$, and all constrains $C_{(v_i,v_{i+1})}$ are satisfied, i.e., $\forall 0 \leq i < t, C_{(v_i,v_{i+1})}(A^t_{v_i}(v_0), A^t_{v_{i+1}}(v_0)) = \mathsf{accept}$.
In the next section, we focus on proving the reduction validity, that is, we show its complexity and soundness satisfy the conditions of Theorem 9.1.

9.2 Validity of the reduction

Complexity Since t and d do not depend on the size of $\mathcal{G}$, the reduction yields a constraint graph that has $|E^t| \leq d^t \cdot |E| = O(|E|)$ edges and alphabet of size $|\hat{\Sigma}| = |\Sigma^{d^t}| = |\Sigma|^{d^t} = |\Sigma|^{O(1)}$. This proves that the reduction is polynomial.

Completeness If $\mathcal{G}$ has a valid assignment A, then it may be used for constructing a valid A^t assignment to $\mathcal{G}^t$ as follows. We define the opinion of a vertex v regarding a value of a vertex u to be the actual value assigned to it, i.e., $A^t_u(v) = A(u)$. Then for every edge $\bar{e}$, the opinions of v_0 and v_t are consistent: $A^t_{v_i}(v_0) = A(v_i) = A^t_{v_i}(v_t)$, and the constraints $C_{(v_i,v_{i+1})}$ are satisfied: $C_{(v_i,v_{i+1})}(A^t_{v_i}(v_0), A^t_{v_{i+1}}(v_0)) = C_{(v_i,v_{i+1})}(A(v_i), A(v_{i+1})) = \mathsf{accept}$. Hence completeness holds.

Soundness Consider the soundness of $\mathcal{G}^t$. We start by reconstructing an assignment $\hat{A}$ to $\mathcal{G}$ from an assignment A^t to $\mathcal{G}^t$. It will help us to prove the soundness of the reduction.

Definition 9.4. Given an assignment A^t to $\mathcal{G}^t$, define $\hat{A} : V \to \Sigma$ as follows: $\hat{A}(v) = \arg\max_{\sigma \in \Sigma} \Pr[\text{a random walk of length } \frac{t}{2} \text{ starting at } v \text{ ends at } w \text{ s.t. } A^t_v(w) = \sigma]$.

By Definition 9.4, we immediately get that $\Pr_w[A^t_v(w) = \hat{A}(v)] \geq \frac{1}{|\Sigma|}$.
Using assignment $\hat{A}$, we formulate the following lemma, which implies soundness of the reduction and, particularly, Theorem 9.1.

Lemma 9.5 (Main). *For every even t and every constraint graph $\mathcal{G}$ over (d,λ)-expander with loops and for every assignment $A^t : V \to \Sigma^{d^t}$, there exists a constant $\beta = \beta(\lambda, d, |\Sigma|)$, such that $s(\mathcal{G}^t, A^t) \geq \beta\sqrt{t}\min\{s(\mathcal{G}, \hat{A}), \frac{1}{t}\}$.*

Lemma 9.5 immediately implies soundness of the reduction:

Lemma 9.6 (Soundness of $\mathcal{G}^t$). *Given β, t as above, then $s(\mathcal{G}^t) \geq \beta\sqrt{t}\min\{s(\mathcal{G}), \frac{1}{t}\}$ holds.*

Proof. By Definition 8.7, we get that for any assignment A, $s(\mathcal{G}, A) \geq s(\mathcal{G})$ (the same for A^t and $\mathcal{G}^t$). Assuming that A^t is the "best" assignment to $\mathcal{G}^t$, and applying Lemma 9.5 to it, we get $s(\mathcal{G}^t) = s(\mathcal{G}^t, A^t) \geq \beta\sqrt{t}\min\{s(\mathcal{G}, \hat{A}), \frac{1}{t}\} \geq \beta\sqrt{t}\min\{s(\mathcal{G}), \frac{1}{t}\}$ □

We start to prove main lemma by introducing new definitions and proving some claims.

Definition 9.7. $F = \{(u,v) \in E | C_{(u,v)}(\hat{A}(u), \hat{A}(v)) = \mathsf{reject}\}$.

Informally, $F \subseteq E$ is a set of edges having constraints that $\hat{A}$ violates. By Definition 8.7, $|F| = |E| \cdot s(\mathcal{G}, \hat{A})$. Clearly,

$$\frac{|F|}{|E|} \geq \min\{s(\mathcal{G}, \hat{A}), \frac{1}{t}\}. \tag{1}$$

Definition 9.8. We say that a path $\bar{e} = (v_0, \ldots, v_t)$ *is hit by* a position i (i.e., by edge (v_{i-1}, v_i)), if $(v_{i-1}, v_i) \in F$, $A^t_{v_{i-1}}(v_0) = \hat{A}(v_{i-1})$, $A^t_{v_i}(v_t) = \hat{A}(v_i)$.

Definition 9.9. $I = \{\frac{t}{2} - \sqrt{\frac{t}{2}}, \ldots, \frac{t}{2} + \sqrt{\frac{t}{2}}\}$.

Definition 9.10. $N(\bar{e}) = |\{i \in I | \bar{e} \text{ is hit by } i\}|$.

Clearly, if $\bar{e}$ is hit by some i, then $C_{\bar{e}}(A^t(v_0), A^t(v_t)) = \mathsf{reject}$.

Claim 11. $\mathrm{Ex}_{\bar{e}}[N(\bar{e})] \geq \Omega(\sqrt{t}) \cdot \frac{|F|}{|E|}$.

Claim 12. $\mathrm{Ex}_{\bar{e}}[(N(\bar{e}))^2] \leq O(\sqrt{t}) \cdot \frac{|F|}{|E|}$.

Fact 9.13. If a random variable $X \geq 0$ is not identical to 0, then $\Pr[X > 0] \geq \frac{(\mathrm{Ex}[X])^2}{\mathrm{Ex}[X^2]}$.

Proof of Fact. Let $\mathbf{1}_{X>0}$ be the indicator random variable for the event $X > 0$. By the Cauchy-Schwartz inequality, $\mathrm{Ex}[X] = \mathrm{Ex}[X \cdot \mathbf{1}_{X>0}] \leq \sqrt{\mathrm{Ex}[X^2]} \times \sqrt{E[\mathbf{1}^2_{X>0}]} = \sqrt{\mathrm{Ex}[X^2]} \times \sqrt{\Pr[X > 0]}$. Then we get $(\mathrm{Ex}[X])^2 \leq \mathrm{Ex}[X^2] \cdot \Pr[X > 0]$. □

Assuming Claim 11 and Claim 12 hold we may prove the main lemma as follows:

Proof of Lemma 9.5.

$$\begin{aligned} s(\mathcal{G}^t, A^t) &\overset{(Definition\ 9.8)}{\geq} \Pr_{\bar{e}\in E^t}[N(\bar{e}) > 0] \\ &\overset{(Fact\ 9.13)}{\geq} \frac{(\mathrm{Ex}[N(\bar{e})])^2}{\mathrm{Ex}[(N(\bar{e}))^2]} \\ &\overset{(Claim\ 11, Claim\ 12)}{\geq} \Omega(\sqrt{t}) \cdot \frac{|F|}{|E|} \\ &\overset{(Equation\ 1)}{\geq} \Omega(\sqrt{t}) \cdot \min\{s(\mathcal{G}, \hat{A}), \frac{1}{t}\}. \end{aligned}$$

□

Now we return to the proof of Claim 11.

Proof of Claim 11. We introduce an indicator random variable $N_i(\bar{e})$ for the event "$\bar{e}$ is hit by i". Then, $N(\bar{e}) = \sum_{i\in I} N_i(\bar{e})$, and $\mathrm{Ex}_{\bar{e}}[N(\bar{e})] = \sum_{i\in I} \mathrm{Ex}_{\bar{e}}[N_i(\bar{e})] = \sum_{i\in I} \Pr_{\bar{e}}[N_i(\bar{e})]$. Let's fix some $i \in I$ and build a random path $\bar{e}$ as follows.

1. Choose edge (v_{i-1}, v_i) randomly.
2. Choose randomly paths of length $i-1$ and $t-i$ starting from vertices v_{i-1} and v_i.
3. Build $\bar{e}$ from these two paths and from the edge (v_{i-1}, v_i).

The edge $\bar{e}$ is randomly chosen, because the graph G is d-regular. Then we get

$$\begin{aligned} \Pr_{\bar{e}}[N_i(\bar{e})] &= \Pr_{\bar{e}}[i \text{ hits } \bar{e}] \\ &= \Pr_{\bar{e}}\left[(v_{i-1}, v_i) \in F \wedge A^t_{v_{i-1}}(v_0) = \hat{A}(v_{i-1}) \wedge A^t_{v_i}(v_t) = \hat{A}(v_i)\right] \\ &\geq \Pr_{\bar{e}}[(v_{i-1}, v_i) \in F] \cdot p_{v_0} \cdot p_{v_t} = \frac{|F|}{|E|} \cdot p_{v_0} \cdot p_{v_t}, \end{aligned}$$

where $p_{v_0} = \Pr_{v_0}[A^t_{v_{i-1}}(v_0) = \hat{A}(v_{i-1})]$, and $p_{v_t} = \Pr_{v_t}[A^t_{v_i}(v_t) = \hat{A}(v_i)]$. Since for $i = \frac{t}{2}$, the following holds:

$$p_{v_0} = p_{v_t} = \Pr_{w}\left[A^t_{v_{\frac{t}{2}}}(w) = \hat{A}(v_{\frac{t}{2}})\right] \geq \frac{1}{|\Sigma|},$$

we get that

$$\Pr_{\bar{e}}[N_{\frac{t}{2}}(\bar{e})] \geq \frac{|F|}{|E|} \cdot \frac{1}{|\Sigma|^2}.$$

Using the fact (not proved here, for a proof see Dinur [2007]) that there is $\epsilon > 0$, such that for any $i \in I$, $\Pr_{\bar{e}}[N_i(\bar{e})] \geq \epsilon \cdot \Pr_{\bar{e}}[N_{\frac{t}{2}}(\bar{e})]$ holds, and that $|I| = \sqrt{2t} + O(1)$ we get

$$\mathop{\mathrm{Ex}}_{\bar{e}}[N(\bar{e})] = \sum_{i \in I} \Pr_{\bar{e}}[N_i(\bar{e})] \geq |I| \cdot \epsilon \cdot \Pr_{\bar{e}}[N_{\frac{t}{2}}(\bar{e})] \geq |I| \cdot \epsilon \cdot \frac{|F|}{|E|} \cdot \frac{1}{|\Sigma|^2} \geq \Omega(\sqrt{t}) \cdot \frac{|F|}{|E|}.$$

□

In the next lecture we will complete the proof of validity of the reduction by providing a proof for Claim 12.

Lecture 10

Parallel Repetition I — Definitions and motivation

March 24th, 2008

Lecturer: Eli Ben-Sasson Scribe: Kineret Ben-Eliyahu and Hanna Fadida

10.1 Validity of the reduction - continue

This lecture we will finish the proof of **??** from the previous lecture, by proving claim **??**:

$$E\left[N^2\left(\overline{e}\right)\right] \leq O\left(\sqrt{t}\right)\frac{|F|}{|E|}$$

Proof. In the previous lecture we defined the interval I as $I = \left[t/2 - \sqrt{t/2}, t/2 + \sqrt{t/2}\right]$ and $F = \{(u,v) \in E | C_{(u,v)}(\hat{A}(u), \hat{A}(v)) = \mathsf{reject}\}$
Let's define $Z\left(\overline{e}\right)$ for a path $\overline{e} = (v_0, \ldots, v_t)$ to be the number of 'bad' edges around the middle of the path:

$$Z\left(e\right) = |\{(v_i, v_{i+1}) \in F | i \in I\}|$$

Let Z_i be the indicator variable of the event "the edge (v_i, v_{i+1}) is in F"'.
Therefore,

$$Z\left(e\right) = \sum_{i \in I} Z_i\left(e\right)$$

.

$$\begin{aligned} E\left[Z^2\left(e\right)\right] &= E\left[\left(\sum Z_i\left(e\right)\right)^2\right] = E\left[\sum_{i,j \in I} Z_i\left(e\right) Z_j\left(e\right)\right] = \sum_{i \in I} E\left[Z_i\right] + 2\sum_{i<j} E\left[Z_i Z_j\right] = \\ &\overset{(*)}{=} |I| \cdot \frac{|F|}{|E|} + 2\sum_{i<j}\left[\left(\frac{|F|}{|E|}\right)^2 + \frac{|F|}{|E|} \cdot \left(\frac{\lambda}{d}\right)^{j-i}\right] \end{aligned}$$

The graph G is a regular graph and therefore, $E\left[Z_i\right] = \frac{|F|}{|E|}$.
$E\left[Z_i Z_j\right]$ is the probablity that both edges are in F.

$$E\left[Z_i Z_j\right] = \Pr\left[Z_i = 1\right] \Pr\left[Z_j = 1 | Z_i = 1\right]$$

.
If G is (d,λ)-expander graph, then random walk which starts in F does it's i-th move in F with probability $\leq \frac{|F|}{|E|} + \left(\frac{\lambda}{d}\right)^i$ and thus $\Pr\left[Z_i = 1\right] \Pr\left[Z_j = 1 | Z_i = 1\right] \leq \frac{|F|}{|E|} \cdot \left(\frac{|F|}{|E|} + \left(\frac{\lambda}{d}\right)^{j-i}\right)$.

$$E\left[Z^2(e)\right] = E\left[\left(\sum Z_i(e)\right)^2\right] = E\left[\sum_{i,j\in I} Z_i(e)\,Z_j(e)\right] = \sum_{i\in I} E\left[Z_i\right] + 2\sum_{i<j} E\left[Z_i Z_j\right] =$$
$$= |I|\cdot\frac{|F|}{|E|} + 2\sum_{i<j}\left[\left(\frac{|F|}{|E|}\right)^2 + \frac{|F|}{|E|}\cdot\left(\frac{\lambda}{d}\right)^{j-i}\right] \leq |I|\cdot\frac{|F|}{|E|} + \frac{|F|}{|E|}\cdot|I|^2\cdot\frac{|F|}{|E|} + 2\frac{|F|}{|E|}\sum_{i<j}\left(\frac{\lambda}{d}\right)^{j-i}$$

Due to the definitions above $|I| = O\left(\sqrt{t}\right)$ and $\frac{|F|}{|E|} \leq \frac{1}{\sqrt{t}}$. In addition we choose $\lambda = \frac{d}{2}$, and therefore, $|I|\cdot\frac{|F|}{|E|} + \frac{|F|}{|E|}\cdot|I|^2\cdot\frac{|F|}{|E|} + 2\frac{|F|}{|E|}\sum_{i<j}\left(\frac{\lambda}{d}\right)^{j-i} \leq O\left(\sqrt{t}\right)\cdot\frac{|F|}{|E|} + \frac{|F|}{|E|}\cdot O\left(\sqrt{t}\right) + 2\frac{|F|}{|E|}\sum_{i<j}2^{i-j} \leq O\left(\sqrt{t}\right)\cdot\frac{|F|}{|E|}$

By definition $N(e) < Z(e)$, and therefore, $E\left[N^2(e)\right] \leq O\left(\sqrt{t}\right)\frac{|F|}{|E|}$.

□

This claim was necessary for the soundness' proof, and finishes phase II.

10.2 Parallel Repetition - Motivation

Recall that we want to show that for every $\epsilon > 0$ there exist an alphbet Σ in a constant size such that

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} q & \leq & 2 & c & = & 1 \\ \Sigma & = & O_\epsilon(1) & s & \geq & 1-\epsilon \\ t(n),\ell(n) & = & f(n)^{O_\epsilon(1)} & & & \end{array}\right).$$

We have already seen in the first lecture (Theorem 1.3) that for every proper complexity function
$f : \mathbb{N}^+ \to \mathbb{N}^+$ and all $\epsilon > 0$,

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} q & \leq & 3 & & & \\ \Sigma & = & \{0,1\} & & & \\ \mathsf{nonadaptive} & & & c & \geq & 1-\epsilon \\ \mathsf{query-type} & & \mathrm{XOR} & s & \geq & \frac{1}{2}-\epsilon \\ t(n),\ell(n) & \leq & \mathrm{poly}(f(n)) & & & \\ r(n) & \leq & \log(\ell(n)) + O(1) & & & \end{array}\right).$$

To prove this we will think of a proof as a game with two players.

10.3 Parallel Repetition - Definitions

Definition 10.1 (2-Prover 1-Round Game)**.** A 2-prover 1-round game G consists of four sets: X, Y, A, B, with probability measure μ on $X \times Y$, and a decision predicate $V : X \times Y \times A \times B \to acc/rej$

A strategy is a pair of functions:

$$\pi_0 : X \times R \to A$$

$$\pi_1 : Y \times R \to B$$

And the value of the game, $\omega(G)$, is defined to be:

$$\omega(G) = \max_{\pi_0,\pi_1} \Pr_{x,y\sim\mu}[V(x, y, \pi_0(x, r), \pi_1(y, r)) = acc]$$

Definition 10.2 (Soundness Error)**.** The Soundness Error is defined to be $e_s = 1 - s$.

One way to decrease the soundness error is Sequential repetition, which decreases the soundness error exponentially, but requires multiple rounds. The solution is *Parallel Repetition*.

Definition 10.3. A n-repeated game, G^n, is the game over $X^n \times Y^n \times A^n \times B^n$ with a decision predicate $V^n(x_1, \ldots, x_n, y_1, \ldots, y_n, a_1, \ldots, a_n, b_1, \ldots, b_n) = acc \Leftrightarrow \bigwedge_{i=1}^{n} V(x_i, y_i, a_i, b_i) = acc$

What is $\omega(G^n)$?

Theorem 10.4 (Fortnow-Sipser-Rompel '88)**.** $\omega(G^n) = (\omega(G))^n$.

This "theorem" turn out to be wrong.
Counterexample: consider following game:

- V chooses $x_0, x_1 \in \{0, 1\}$, and sends x_i to prover P_i
- Prover P_0 responds with (a_0, a_1)
- Prover P_1 responds with (b_0, b_1)
- V accepts iff $(a_0, a_1) = (b_0, b_1)$ and $a_1 = x_{a_0}$

i.e. the provers reply with "Prover P_{a_0} got a_1". The value of the game is $\frac{1}{2}$, for the strategy in which P_0 replies $(0, x_0)$, and P_1 replies $(0, b_1)$. With probability $\frac{1}{2}$, P_1 "guess" right, and so $\omega(G) = \frac{1}{2}$.

What about $\omega\left(G^2\right)$?

Suppose provers choose the following strategy:

$$P_0\text{: } \left(0, x_0^{(1)}\right)\left(1, x_0^{(1)}\right).$$
$$P_1\text{: } \left(0, x_1^{(2)}\right)\left(1, x_1^{(2)}\right).$$

The provers win iff $x_0^{(1)} = x_1^{(2)}$, which happens with probability $\frac{1}{2}$. Hence, $\omega\left(G^2\right) \geq \frac{1}{2}$. However,

Claim 5. $\forall \omega\left(G^n\right) = 2^{-n/2}$.

So, the value of the game goes down exponentially.

Theorem 10.6. *(*Parallel Reptition Theorem,Raz [1998][Raz '95]) For any game G, there exist a constant $\alpha_G > 0$ (that depends only on the alphabet size of the provers' answers, $|A|, |B|$, and on $\omega(G)$), such that $\omega\left(G^n\right) \leq (\alpha_G)^n$.

Next lecture we will prove the theorem above.

LECTURE 11

PARALLEL REPETITION II — SKETCH OF PROOF

MARCH 31st, 2008

LECTURER: Eli Ben-Sasson SCRIBE: Kineret Ben-Eliyahu and Hanna Fadida

11.1 Proof of Parallel Repetition Theorem

In the previous lecture we introduced the Parallel Repetition Theorem by Raz (Theorem 10.6). Today we are going to prove a new versia of this theorem that was introduced by Holenstein:

Theorem 11.1. *(*[Holenstein '07]) if $\omega(G) = 1-\delta$ then $\omega(G^n) \leq 2^{-c\delta^3 n/2}$ (when c depends on the alphabet, $c \sim \frac{1}{log(|A||B|)}$).

Note that improving of the paramaeters, e.g, $\omega(G^n) \leq 2^{-\delta n}$, will lead to a proof for a very well known assumption - "unique game conjuction".
Let W_i be the event of winning the $i-th$ game, i.e, $V(x_i, y_i, a_i, b_i) = acc$.

Lemma 11.2. *(Main Lemma)*
For any alphabet A, B, there exists a constant $c > 0$, such that for any δ, and for all $k \leq c\delta^2 n$, there exists $j > k$, such that if $\omega(G) = 1-\delta$ then:

$$\Pr[W_j | W_1 \cdots W_k] \leq 1 - \delta/2. \tag{2}$$

The above lemma implies Theorem 11.1:

$$\begin{aligned}\Pr\left[\bigwedge_{i=1}^{n} W_i\right] &\leq \Pr[W_1]\Pr[W_2|W_1]\cdots\Pr[W_n|W_1\cdots W_{n-1}] \leq (1-\delta)\underbrace{(1-\delta/2)\cdots(1-\delta/2)}_{c\delta^2 n}\cdot 1\cdots 1 \\ &\leq (1-\delta)(1-\delta/2)^{c\delta^2 n} \leq e^{-c\delta^3 n/2}\end{aligned}$$

Proof. of Lemma 11.2. We assume in contradiction that there is a provers' stategy violating 2 and will use it to succeed in the original game, with probability more than $1-\delta$, in contradiction to assumption that $\omega(G) = 1-\delta$.
We define $W = \bigwedge_{i=1}^{k} W_i$.

If to any $j > k$

$$\Pr\left[W_j | \bigwedge_{i=1}^{k} W_i\right] > 1 - \delta/2$$

then:

Claim 3. *There exists a winning transcript of the first k rounds, $\bar{a}\bar{b} \in A^k \times B^k$, such that:*

1. $\Pr\left[t = \bar{a}\bar{b} | W\right] \geq \frac{1}{100} \left(|A| \, |B|\right)^{-k}$
2. *For at least $\frac{1}{100}$ percents of the release $j - ies$ (from $k+1, \ldots, n$)* $\Pr\left[W_j | W \wedge t = \bar{a}\bar{b}\right] \geq 1 - 3/4\delta$

Proof. Let us define $e_j = \Pr\left[W_j | W\right]$.
We say that a transcript t is **bad** if $\left|\left\{j | e_j|_t > 1 - \frac{3}{4}\delta\right\}\right| < \frac{1}{100}(n-k)$.

Claim 4. $\Pr\left[t \text{ is bad } | W\right] \leq \frac{99}{100}$

Proof. We assume in contradiction that $\Pr\left[t \text{ is bad } | W\right] > \frac{99}{100}$, then, we look at $\frac{\sum_{j>k} e_j}{n-k}$, in one hand:

$$\frac{(n-k)(1-\delta/2)}{n-k} < \frac{\sum_{j>k} e_j}{n-k}$$

because the contradiction assumption of the previous claim 11.1.
For the other hand:

$$\begin{aligned}
\frac{1}{n-k} \cdot \sum_{j>k} e_j &\geq \Pr\left[t \text{ is bad }\right] \left[\frac{99}{100}\left(1 - \frac{3}{4}\delta\right) + \frac{1}{100} \cdot 1\right] + \left(1 - \Pr\left[t \text{ is bad }\right]\right) \cdot 1 \\
&\geq \frac{99}{100}\left[\frac{99}{100}\left(1 - \frac{3}{4}\delta\right) + \frac{1}{100}\right] \leq \left(\frac{99}{100} - \frac{99}{100} \cdot \frac{3}{4}\delta\right) + \frac{1}{100} < 1 - \frac{\delta}{2}
\end{aligned}$$

□

Because of the calim above, and the fact that there is only $|A|^k \times |B|^k$ possible transcripts, there must be a "good" transcript t that happens with probability greater than $\frac{1}{|A|^k \times |B|^k} \cdot \Pr\left[t \text{ is good } | W\right] \geq \frac{|A|^k \times |B|^{-k}}{100}$, and the proof of claim Claim 3 is completed. □

Now we take a transcript $t = \overline{a}, \overline{b}$ like this, which happens with probability greater than $\frac{|A|^k \times |B|^{-k}}{100}$. Moreover, the winning probability for many of the $W_j - ies$ $(j > k)$, while $\overline{a}, \overline{b}$ happens, is greater than $1 - \delta$ (in fact it's greater than $1 - \frac{3}{4}\delta$).
We will look at the distribution on $x_{k+1}, \ldots, x_n, y_{k+1}, \ldots, y_n$ conditions on $W \wedge t$:
$\widetilde{X}_{k+1}, \ldots, \widetilde{X}_n, \widetilde{Y}_{k+1}, \ldots, \widetilde{Y}_n$. This is the distribution of the last queries that is conditioned in winning on the first k games, which happpens by answering t in the k queries.

Lemma 11.5. *Let $U = U_1 \times \ldots \times U_n$ be a product distribution, and let $\widetilde{U} = \widetilde{U}_1 \times \ldots \times \widetilde{U}_n$ be the distribution of U conditioned on some event that happens with probability at least 2^{-d}. Then,*

$$\frac{1}{n} \sum_j \Delta \left(U_j, \widetilde{U} j \right) \leq \sqrt{d/n}$$

□

Lecture 12

Towards 3-query PCPs with optimal soundness

April 7th, 2008

Lecturer: Eli Ben-Sasson Scribe: Nathaniel Cohen and Yonathan Touati

In this final lecture we shall sketch the proof of Theorem 1.3. For a detailed proof see, e.g., Khot [2004]. We start with a restatement of the theorem we wish to obtain.

Theorem 1.3 (restated). *For every proper complexity function $f : \mathbb{N}^+ \to \mathbb{N}^+$ and all $\epsilon > 0$,*

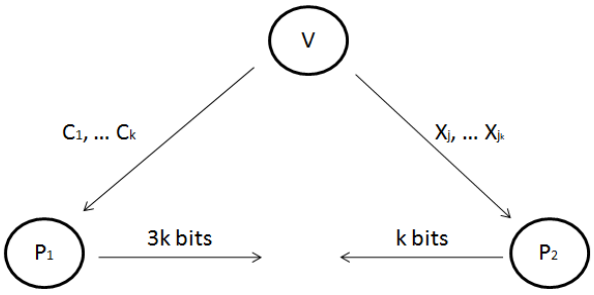
$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} q & \leq & 3 & & & \\ \Sigma & = & \{0,1\} & & & \\ \text{nonadaptive} & & & c & \geq & 1-\epsilon \\ \text{query}-\text{type} & & XOR & s & \geq & \frac{1}{2}-\epsilon \\ t(n), \ell(n) & \leq & \text{poly}(f(n)) & & & \\ r(n) & \leq & \log(\ell(n)) + O(1) & & & \end{array}\right).$$

From previous lectures, we know something very close to Theorem 1.3; The basic PCP construction from Corollary 8.4 plus the parallel repetition theorem gives that for all $\epsilon > 0$ there exists a constant $c_\epsilon > 0$ and an alphabet Σ of size c_ϵ such that

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} q & = & 2 & c & = & 1 \\ \Sigma & & & s & \geq & 1-\epsilon \\ t(n), \ell(n) & = & f(n)^{O(1)} & & & \end{array}\right).$$

To obtain Theorem 1.3 we need to reduce the alphabet size to 2, increase the query complexity to 3 and use only XOR-constraints in our verification process. We shall do this by encoding symbols in the "large" alphabet Σ by a special error correcting code, called the *long code*.

Inspecting the application of parallel repetition to our basic PCP from Corollary 8.4, recall our verifier can be graphically described via the following figure.



This implies that the following gap-problem is **NP**-hard. Recall the definition of a constraint graph (Definition 8.5).

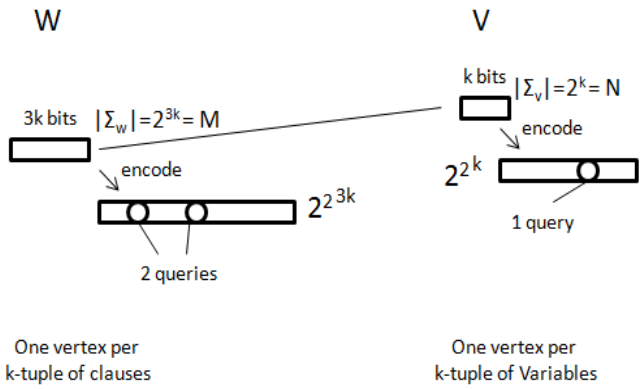
Definition 12.1 (GAP_ϵ Label Cover)**.** An instance of the language Label Cover (LC) is a constraint graph $\mathcal{G} = \{G, \Sigma, \mathcal{C}\}$ where

- G is bipartite. Let W, V denote the partition of its vertices.
- Σ is partitioned into Σ_W and Σ_V.
- For every constraint $C_{(v,w)} \in \mathcal{C}$, the set of assignments (σ, σ') that satisfy $C_{(v,w)}$ is a subset of $\Sigma_V \times \Sigma_W$. In other words, $C_{(v,w)}$ can be considered a constraint from $\Sigma_V \times \Sigma_W$ to $\{\mathsf{accept}, \mathsf{reject}\}$.

For $\epsilon > 0$ let GAP_ϵ Label Cover is the promise problem over Label Cover instances defined by

- YES $= \{\mathcal{G} | s(\mathcal{G}) = 0\}$.
- NO $= \{\mathcal{G} | s(\mathcal{G}) \geq 1 - \epsilon\}$.

The way we shall encode symbols of Σ_V, Σ_W can be pictorially described as follows.



We shall define a verifier that, given encodings for symbols of Σ_V, Σ_W as above, will make 3 (bit) queries, take their XOR and based on this value output $\mathsf{accept}/\mathsf{reject}$.

12.1 The long code

Next, we define the code that is used in the 3-query PCP of Theorem 1.3. Recall the Hadamard code (Definition 3.3).

Definition 12.2 (Long code)**.** The *long code* of an N-symbol alphabet is defined by the encoding $L_N : [N] \to \mathbb{F}_2^{2^N}$,

$$L(i) = \text{HADAMARD}_N(e_i), \text{ for } i \in [N],$$

where $e_i \in \mathbb{F}_2^N$ is the vector that has a 1 in the i position and 0 everywhere else.

Notice the codewords of the long code are a subset of the N-dimensional Hadamard code, i.e., $L_N \subset \text{HADAMARD}_N$. Furthermore, since $k = \log N$ is the number of bits required to describe a message, the long code maps k-message bits to $2^N = 2^{2^k}$ codeword-bits, justifying its name — the *long* code. Finally, the long code is not a linear code because the set of codewords is not closed under addition. Indeed, $L(i) + L(j)$ is not a codeword of L_N.
It turns out that using the long code to encode symbols in Σ_W, Σ_V one can obtain Theorem 1.3. We do not have time to show details in this course and refer the reader to Khot [2004] for details.

Homework Assignment 1

Published on 4.2.2008
Due by 11.2.2008

1. To answer this question, recall lecture 2. MAX3SAT is an optimization problem over CNF formulas with exactly 3 literals in each clause. Given such a 3CNF instance ϕ, let $OPT(\phi)$ be the maximal number of clauses that can be simultaneously satisfied by an assignment to the variables.

   - Find a trivial $\frac{7}{8}$-approximation algorithm for MAX3SAT.
   - Prove: If for some $\epsilon > 0$ there exists a polynomial time algorithm that is a $(\frac{7}{8} + \epsilon)$-approximation for MAX3SAT, then $\mathbf{P} = \mathbf{NP}$.

2. Prove: If $w : F_2^k \to F_2$ satisfies $w_a + w_b = w_{a+b}$ for all $a, b \in F_2^k$, then w is a codeword of the k-dimensional Hadamard code. Hint: Basic linear algebra.

3. In this question we shall prove that any linear function can be locally decoded from the Hadamard code with two queries. Let $E : F_2^k \to F_2^{2^k}$ be the encoding function of the Hadamard code. Prove: There exists a *decoder*, i.e., a randomized machine D with oracle access to a word $w \in F_2^{2^k}$ such that for any input $a = (a_1, \ldots, a_k) \in F_2^k$, the decoder makes two queries to w and the following guarantee holds: If w is δ-close to $E(m)$ for some $m \in F_2^k$, then

$$\Pr_R \left[D^w[a; R] = \sum_{i=1}^{k} a_i m_i \right] \geq 1 - 2\delta,$$

   where $D^w[a; R]$ denotes the (one bit) output of D on oracle w, input a and random coins R.

Homework Assignment 2

Published on 12.2.2008
Due by 18.2.2008

This assignment ties a couple of loose ends in the proof of the exponential length PCP Theorem. The first question completes the analysis of the Hadamard tester. The last three questions provide a formal proof of the following statement, discussed in the past two lectures and stated there as Theorem 3.1.

Theorem 1. *There exists $\epsilon > 0$ such that for any proper $f : \mathbb{N}^+ \to \mathbb{N}^+$,*

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|lcl} q & = & O(1) & & & \\ \Sigma & = & \mathbb{F}_2 & & & \\ \mathsf{nonadaptive} & & & c & = & 1 \\ t(n) & \leq & \mathrm{poly}(f(n)) & s & \geq & \epsilon \\ \ell(n) & \leq & 2^{f^2(n)} & & & \\ r(n) & \leq & O(f^2(n)) & & & \end{array}\right).$$

The notation for the list of restrictions is the same as in Theorem 1.3 and $\mathbb{F}_2$ denotes the two-element field.

1. Carefully read and summarize the soundness analysis of the 3-query tester for the Hadamard code, stated as Lemma 2.11 in Lecture 2 of Ben-Sasson [Fall 2005]. Pay particular attention to the proof of Claim 14 (which we didn't complete in class).

2. An instance of MATRIXSAT of size t is a collection of at most t constraints $\phi_1, \phi_2, \ldots$ over a set of variables $y_{ij}, i, j \in [t] = \{1, \ldots, t\}$. Each constraint ϕ_r involves at most two variables and is an $\mathbb{F}_2$-linear constraint, i.e., ϕ_r is of the form $y_{i,j} + y_{k,l} + b = 0$ for some $i, j, k, l \in [t], b \in \mathbb{F}_2$ and addition is done modulo 2. An instance ϕ is said to be *satisfied* by $\beta \in \mathbb{F}_2^{t \times t}$ if *(i)* β satisfies all constraints and *(ii)* $\beta = \alpha \cdot \alpha^T$ for some $\alpha \in \mathbb{F}_2^t$, or in other words, $\beta_{ij} = \beta_{ii} \cdot \beta_{jj}$ for all $i, j \in [t]$. Let MATRIXSAT be the language consisting of all satisfiable instances, i.e., instances that can be satisfied by some β.

   Prove: There exists a $\mathrm{poly}(f(n))$-time reduction from any language $L \in \mathbf{NTIME}\left(f(n)\right)$ to MATRIXSAT, where an instance of L of size n is reduced to an instance of MATRIXSAT of size $f(n)$.

3. Prove: There exists a verifier V that on input ϕ, an instance of MATRIXSAT of size t and access to an oracle π of length 2^{t^2} has the following properties.

   - **Operation:** V tosses $O(t)$ coins, runs in time $\mathrm{poly}\left(t^2\right)$, makes 4 nonadaptive queries to π and outputs either accept or reject.
   - **Completeness:** If π is the Hadamard encoding of $\beta \in \mathbb{F}_2^{t^2}$ and β satisfies ϕ then $\Pr_R[V^\pi[\phi; R] = \mathsf{accept}] = 1$.

- **Completeness:** If π is the Hadamard encoding of $\beta \in \mathbb{F}_2^{t^2}$ and β does not satisfy ϕ then $\Pr_R[V^\pi[\phi; R] = \mathsf{reject}] \geq 1/4$.

4. Give a formal proof of Theorem 1. Use *(i)* the local tester for the Hadamard code presented in Lecture 2, *(ii)* the local decoder for Hadamard codes from Homework assignment 1 and *(iii)* the answers to the previous questions.

Homework Assignment 3

Published on 18.2.2008
Due by 25.2.2008

The goal of this homework is to get familiarized with *(i)* the notion of a PCPP and *(ii)* the PCPP Composition Theorem, both discussed in Lecture 5.

1. Prove, using the exponential length PCP discussed in Lectures 3 – 4: There exists $\epsilon > 0$ such that

$$\text{PAIR-SAT} \in \mathbf{PCPP}\left(\begin{array}{lcl|lcl} q & = & O(1) & & & \\ \Sigma & = & \mathbb{F}_2 & c & = & 1 \\ \ell(n) & = & 2^{n^2} & s(n,\delta) & \geq & \epsilon \cdot \delta \\ r(n), t(n) & = & O(n^2) & & & \end{array}\right)$$

2. Recall the Composition Theorem stated in class. We say the soundness function $s : \mathbb{N}^+ \times [0,1] \to [0,1]$ is *convex* if for every n the function $s(n,\cdot) : [0,1] \to [0,1]$ is convex. Let $d(n)$ denote the size of the decision circuit generated by the PCPP-verifier on input of size n (all other parameters are as defined in Theorem 1.3). The Composition Theorem says that for convex soundness function s,

$$\text{If PAIR-SAT} \in \mathbf{PCPP}\left(\begin{array}{l|l} q(n) & \\ d(n) & \\ r(n) & c = 1 \\ \ell(n) & s(n,\delta) \\ t(n) & \end{array}\right),$$

$$\text{Then PAIR-SAT} \in \mathbf{PCPP}\left(\begin{array}{l|l} q(d(n)) & \\ d(d(n)) & \\ r(n) + r(d(n)) & c = 1 \\ \ell(n) + 2^{r(n)} \cdot \ell(d(n)) & s(d(n), s(n,\delta)) \\ t(n) + t(d(n)) & \end{array}\right).$$

Use the composition theorem and the previous question to fill in the missing parameters:

$$\text{If PAIR-SAT} \in \mathbf{PCPP}\left(\begin{array}{lcl|lcl} q(n), d(n), r(n) & = & O(\log n) & c = 1 & & \\ \ell(n), t(n) & = & n^{O(1)} & s(n,\delta) & \geq & (1-\epsilon)\delta \end{array}\right),$$

$$\text{Then PAIR-SAT} \in \mathbf{PCPP}\left(\begin{array}{lcl|lcl} q(n) & = & O(1) & & & \\ d(n) & = & ? & & & \\ r(n) & = & ? & c = 1 & & \\ \ell(n) & = & ? & s(n,\delta) & \geq & ? \\ t(n) & = & ? & & & \end{array}\right).$$

Homework Assignment 4

Published on 3.3.2008
Due by 10.3.2008

The purpose of this assignment is to complete the proof of the following theorem. To complete this assignment you may use [Ben-Sasson, Fall 2005, Lecture 10].

Theorem 1.

$$\mathbf{NTIME}\left(f(n)\right) \subseteq \mathbf{PCP}\left(\begin{array}{lcl|ll} \ell & \leq & f^2(n)\cdot \mathrm{polylog}(f(n)) & & \\ q & = & O(1) & c=1 & \\ t & \leq & f^{O(1)}(n) & s(n) & \geq \ 1/\mathrm{polylog}(f(n)) \\ \Sigma & = & \mathbb{F}_2 & & \end{array}\right).$$

First we prove that the language ACSP, defined below, is $\mathbf{NTIME}\left(f(n)\right)$-complete. This language is an analog of MATRIXSAT that was introduced in Homework assignment 2. Then we use the PCPP for RS- and vanishing RS-codes to construct PCPPs for the language ACSP, completing the proof of Theorem 1. Details follow.
An instance ψ of size t of the language DOMINO over alphabet Σ is a collection of constraints

$$\psi = \{C_{ij} : \Sigma^3 \to \{\mathsf{accept}, \mathsf{reject}\}\}_{i,j\in[t]}.$$

An assignment $A : [t^2] \to \Sigma$ is said to *satisfy* ψ if for all $i, j \in [t-1]$ we have

$$C_{ij}(A(it+j), A(it+j+1), A((i+1)t+j)) = \mathsf{accept}.$$

Our starting point is the following Theorem (for a proof, see Papadimitriou [1994]).

Theorem 2. *There exists an alphabet $\Sigma, |\Sigma| = d$ such that the language* DOMINO *over Σ, which contains all satisfiable* DOMINO *instances, is complete for* $\mathbf{NTIME}\left(f(n)\right)$ *under quadratic time reductions.*

We reduce DOMINO to its algebraic version, defined next.

Definition 3 (Algebraic CSP (ACSP))**.** An instance of size t of ACSP is a tuple $\phi = (\mathbb{F}, H, L_1, L_2, L_3, C^0, C^1)$ where

- $\mathbb{F}$ is a finite field of characteristic 2 and $100dt^2 < |\mathbb{F}| \leq 200dt^2$, where d is the constant from Theorem 2.
- $H \subset \mathbb{F}, |H| = t^2$.
- L_1, L_2, L_3 are linear functions from $\mathbb{F}$ to $\mathbb{F}$, i.e. $L_i : \mathbb{F} \to \mathbb{F}$ is given by $L_i(x) = a_i x + b_i$.
- C^0 is a univariate polynomial of degree at most d.
- C^1 is a polynomial in four variables, $C^1(x, y_1, y_2, y_3)$ where $\deg_x(C^1) \leq |H|$ and $\deg_{y_i}(C^1) \leq d$.

An assignment to ϕ is a univariate polynomial $A, \deg(A) \leq |H|$. We say that A satisfies ϕ iff

- $C^0(A(h)) = 0$ for all $h \in H$.
- $C^1(h, A(L_1(h)), A(L_2(h)), A(L_3(h)) = 0$ for all $h \in H$.

The language ACSP contains all satisfiable instances.

1. Prove: there is a polynomial time reduction from DOMINO to ACSP sending instance ψ of DOMINO of size t to an instance ϕ of ACSP of size t. Hints:
   - Let $H = \{\omega^1, \omega^2, \ldots, \omega^{t^2}\}$ where ω generates the multiplicative group $\mathbb{F}^*$.
   - Map $[t^2]$ to H.
   - Map Σ to an arbitrary subset of $\mathbb{F}$.
   - Map accept to 0 and reject to 1.
   - Define C^0 to be the polynomial that vanishes only on Σ (this polynomial will be used to check that an assignment A evaluates to Σ on every point in H).
   - Define a 3-variate polynomial $\hat{C}_{ij}$ that agrees with the constraint C_{ij} on Σ^3 and has degree at most d in each of its three variables. You may use the fact that for any function $g : \Sigma^3 \to \mathbb{F}$ there exists a trivariate polynomial of degree at most $|\Sigma|$ in each variable that agrees with g on Σ^3. (No need to prove this fact.) The polynomial $\hat{C}_{ij}$ is called the *low degree extension* of the function C_{ij}.
   - Use the polynomial δ_{ij}^H defined by

$$\delta_{ij}^H(x) = \begin{cases} 1 & x = \omega^{it+j} \\ 0 & x \in H \setminus \{\omega^{it+j}\} \end{cases}$$

     to "glue" the constraints $\hat{C}_{ij}$ into one big constraint, namely, the polynomial C^1.

2. Complete the proof of Theorem 1, by providing a PCP verifier for instances of ACSP. Use the following PCPPs, discussed in class:

$$\begin{matrix}\textsc{pair-binary-RS},\\ \textsc{pair-binary-VRS}\end{matrix} \in \mathbf{PCPP} \left(\begin{array}{lcl|lcl} \ell & \leq & n \cdot \mathrm{polylog}(n) & & & \\ q & = & O(1) & c = 1 & & \\ t & \leq & n^{O(1)} & s(n,\delta) & \geq & \delta/\mathrm{polylog}(n) \\ \Sigma & = & \mathbb{F} & & & \end{array} \right).$$

1 Bibliographical notes

The long code was introduced in Bellare et al. [1998], where it was first analyzed in the context of high-soundness PCPs. The proof of Theorem 1.3 which uses the parallel repetition theorem and Fourier analysis of the long code appeared in Håstad [1997].

Homework Assignment 5

Published on 10.3.2008
Due by 17.3.2008

The purpose of this homework is to fill in some of the details of the gap amplification proof of the PCP Theorem and along the way familiarize ourselves with constraint and expander graphs.

1. This question shows that we may move from constant query complexity to query complexity 2 without a great loss in other parameters. Prove:

$$\mathbf{PCP}\left(\begin{array}{l|l} \ell(n) & \\ r(n) & \\ q(n) & c=1 \\ |\Sigma|=a & s(n) \\ \textsf{nonadaptive} & \\ \vdots & \end{array}\right) \subseteq \mathbf{PCP}\left(\begin{array}{l|l} \ell(n)+2^{r(n)} & \\ r(n)+\lceil \log q(n) \rceil & \\ q=2 & c=1 \\ |\Sigma| \leq a^q & s(n)/q \\ \vdots & \end{array}\right).$$

2. Let Gap-CG$_d(\Sigma, s(n))$ be the promise problem Gap-CG$(\Sigma, s(n))$ restricted to instances that are d-regular graphs. Prove: There exists integer d and constant $c > 1$ such that the following holds. There is a polynomial time, linear size, reduction from Gap-CG$(\Sigma, s(n))$ to Gap-CG$_d(\Sigma, s(n)/c)$. Hints:

   - Replace each vertex of degree $m > d$ with $(d-1, (d-1)/100)$ expander with m vertices.
   - Assume the existence and efficient construction of such expanders.
   - Use the following property of a (d, λ)-expander $G = (V, E)$, stated in class: For every $S \subset V, |S| \leq |S|/2$ we have

$$|E(S, \overline{S})| \geq \frac{d-\lambda}{2} \cdot |S|,$$

     where $E(S, \overline{S})$ is the set of edges with one vertex in S and the other not in S.

References

1. Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998. ISSN 0004-5411.

2. Sanjeev Arora and Shmuel Safra. Probabilistic Checking of Proofs: A New Characterization of NP. *J. ACM*, 45(1):70–122, 1998.

3. László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 21–32, New York, NY, USA, 1991. ACM Press. ISBN 0-89791-397-3.

4. Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs, and nonapproximability — towards tight results. *SIAM Journal on Computing*, 27(3):804–915, June 1998.

5. Eli Ben-Sasson. From error correcting codes to inapproximability probabilistically checkable proofs. Course notes, Fall 2005. URL http://www.cs.technion.ac.il/~eli/courses/PCP_Fall_2005/.

6. Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Robust pcps of proximity, shorter pcps and applications to coding. In *Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing (STOC-04)*, pages 1–10, New York, June 13–15 2004. ACM Press.

7. Eli Ben-Sasson and Madhu Sudan. Short PCPs with poly-log rate and query complexity. In *STOC*, pages 266–275, 2005.

8. Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 73–83, New York, NY, USA, 1990. ACM Press. ISBN 0-89791-361-2.

9. Dinur. The PCP theorem by gap amplification. volume 54, 2007.

10. Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP-theorem. In *FOCS*, pages 155–164, 2004. URL http://csdl.computer.org/comp/proceedings/focs/2004/2228/00/22280155abs.htm.

11. Funda Ergün, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *J. Comput. Syst. Sci.*, 60(3):717–751, 2000.

12. Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, 1996. ISSN 0004-5411.

13. Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, June 1981.

14. Johan Håstad. Some optimal inapproximability results. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1997. ACM Press. ISBN 0-89791-888-6.

15. Subhash Khot. Probabilistically checkable proofs and hardness of approximation - course notes, 2004. URL http://www-static.cc.gatech.edu/~khot/pcp-course.html.

16. J. Kilian. A note on efficient zero-knowledge proofs and arguments. In N. Alon, editor, *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 723–732, Victoria, B.C., Canada, May 1992. ACM Press. ISBN 0-89791-512-7.

17. A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988. ISSN 0209-9683.

18. G. A. Margulis. Explicit constructions of expanders. *Problemy Peredači Informacii*, 9(4):71–80, 1973.

19. Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000. URL http://dx.doi.org/10.1137/S0097539795284959.

20. Ryan O'Donnell. A history of the PCP theorem. Course notes on the PCP Theorem and Hardness of Approximation, Autumn 2005. URL http://www.cs.washington.edu/education/courses/533/05au/pcp-history.pdf.

21. Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.

22. Radhakrishnan and Sudan. On dinur's proof of the PCP theorem. *BAMS: Bulletin of the American Mathematical Society*, 44, 2007.

23. Ran Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, June 1998.

24. O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 3, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0850-2.

25. Mario Szegedy. Many-valued logics and holographic proofs. In Jiří Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Proceedings of the 26th International Colloquium on Automata, Languages and Programming, ICALP'99 (Prague, Czech Republic, July 11-15, 1999)*, volume 1644 of *LNCS*, pages 676–686. Springer-Verlag, Berlin-Heidelberg-New York-Barcelona-Hong Kong-London-Milan-Paris-Singapore-Tokyo, 1999. URL http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743&volume=1644&spage=676.