**6.841 Advanced Complexity Theory** April 25, 2005

# Lecture 21

*Lecturer: Madhu Sudan* *Scribe: Paul Valiant*

## 1 Outline

Today we finish the overview of Dinur's combinatorial proof of the PCP theorem.

Recall the problem max-$k$-SAT-$\Sigma$, an instance of which takes the following form.

- We have $n$ variables $x_1, ..., x_n$, taking values in $\Sigma$.
- We have $m$ constraints $C_1, ..., C_m$, each constraint $C_i$ acting on at most $k$ variables via an arbitrary predicate $P : \Sigma^k \to \{0, 1\}$.

The task is to find an assignment to the variables that maximizes the number of satisfied constraints.

Recall from last lecture, that the PCP theorem is equivalent to the statement that one can transform SAT problems $\phi$ into max-$k$-SAT-$\Sigma$ problems $\phi'$ that are robust in the following sense:

There exists a constant $\epsilon$ such that

- If $\phi$ is satisfiable, $\phi'$ is (perfectly) satisfiable.
- If $\phi$ is not satisfiable, then at least $\epsilon$ fraction of the clauses of $\phi'$ must be unsatisfied for any assignment to the variables of $\phi'$.

For a problem $\phi$, denote by UNSAT($\phi$) the minimum nonzero fraction of clauses of $\phi$ that may be unsatisfied. The idea is that if we can transform any 3SAT problem into a problem $\phi'$ with UNSAT($\phi'$) $\geq \epsilon$, then we can *probabilistically* test a claimed assignment to $\phi'$ by testing a random clause; this test would have soundness $1 - \epsilon$. Thus since 3SAT is NP-complete, we can construct such PCPs for any problem in NP.

We note that for a SAT problem $\phi$ with $m$ clauses, the unsatisfiability gap UNSAT($\phi$) is at least $\frac{1}{m}$. The main idea of Dinur's proof is a technique to *amplify* this fraction to a fixed constant $\epsilon$. This takes the form of the following lemma.

**Lemma 1 (main)** *There exists a transformation $T$ mapping max-2-SAT-$\Sigma$ into itself with the following properties:*

- *$T$ increases the number of variables and clauses by an at most linear factor.*
- UNSAT(T($\phi$)) $\geq \min\{\epsilon, 2\text{UNSAT}(\phi)\}$.

Iteratively applying this lemma $\log m$ times will increase UNSAT($\phi$) to a constant $\epsilon$, while increasing the size of $\phi$ by at most polynomial factor.

The proof of this lemma is divided into two sub-lemmas. See the previous lecture notes for details.

**Lemma 2 (Lemma 1)** *There is a reduction $T_1$ from max-2-SAT-$\Sigma$ to max-2-SAT-$\Sigma^l$ for some $l$ such that the unsatisfiability gap becomes "much" larger.*

**Lemma 3 (Lemma 2)** *There is a reduction $T_2$ from max-2-SAT-$\Gamma$ into max-2-SAT-$\Sigma$, that possibly reduces the unsatisfiability gap, but by a fixed constant that is independent of $\Gamma$.*

In turn, Lemma 3 follows from the following two smaller lemmas.

**Lemma 4 (Lemma 2a)** *There is a reduction $T_{2a}$ from max-2-sat-$\Gamma$ to max-k-SAT-$\{0,1\}$, for some $k$.*

**Lemma 5 (Lemma 2b)** *There is a reduction $T_{2b}$ from max-k-SAT-$\{0,1\}$ to max-2-SAT-$\Sigma$.*

Lemma 5 was shown last time. Here we outline the proofs of Lemmas 2 and 4.

## 2 Gadgets

The proofs of these lemmas are going to take the form of "gadgets". The general form of such a gadget (for our purposes) is the following.

We are given a constraint satisfaction problem $\phi$ with variables $\{x_i\}$ and clauses $\{C_j\}$, and wish to transform the problem so as to meet some specified criteria.

We do this via an encoding function $E$, that transforms $\phi$ to $\phi'$ as follows:

- For each variable $x_i$, $E$ maps $x_i$ into a set of variables $x'_{i,1}, ..., x'_{i,t}$, for some $t$.
- For each constraint $C_j$, $E$ introduces auxiliary variables $y_{j,1}, ..., y_{j,u}$ for some $u$.
- Then for $E$ maps each constraint $C_j$ into a set of new constraint $C'_{j,1}, ..., C'_{j,v}$ on some subset of the new variables $\{x', y\}$.

The encoding function $E$ must preserve satisfiability: if $\phi$ is satisfiable, then so is $\phi'$. Furthermore, to relate the satisfaction of clauses of $\phi'$ back to those of the original problem, we introduce a *decoding* function $D$, that will *decode* variable assignments of $\phi'$ to variable assignments of $\phi$ in such a way that if a large fraction of the clauses $C'_{j,1}, ..., C'_{j,v}$ are satisfied, then the $C_j$ will be satisfied on the decoded assignment $D(x'_{i,1}, ...)$.

Such gadgets are fundamental to the following proofs.

## 3 Proof of Lemma 2a

Recall that we are trying to reduce max-2-SAT-$\Gamma$ to max-$k$-SAT-$\{0,1\}$. When constructing an encoding $E$, as described above, we clearly need to represent a variable $x_i$ in $\Gamma$ by at least $\log|\Gamma|$ binary variables $x'_{i,k}$. We however choose to use many more than $\log|\Gamma|$ variables.

Recall that in the reduction we are constructing, we aim to preserve (up to linear factors) the unsatisfiability gap. In other words, if a high percentage of the new constraints $C'$ are satisfied, then we want a guarantee that a comparably high percentage of the original constraints $C_j$ are satisfied.

We are motivated by the PCP construction to express the probabilistically checkable constraints in terms of tables of function evaluations. The gadget is constructed as follows.

For each variable $x_i$, construct variables $x'_{i,f}$ for *every* function $f : \Gamma \to \{0,1\}$. We will later construct constraints that will let an assignment $\{x'_{i,f}\}$ be interpreted as the result of $f$ being applied to the *original* variable $x_i$.

Recall that each constraint $C_j$ consists of a predicate $P(x_{i_1}, x_{i_2})$ operating on two of the (original) variables. For each such constraint $C_j$, we construct auxiliary variables $y_{j,g}$ for every function $g : \Gamma \times \Gamma \to \{0,1\}$. As above, we will construct constraints so that if a high fraction of the constraints on the transformed variables $x'_*, y_*$ are satisfied then the values $y_{j,g}$ may be correctly interpreted as the values of $g$ on a pair $(a,b)$, which will be seen to satisfy the $j$th constraint of the original problem.

The constraint $C_j$ is transformed into a set of constraints $C'_{j,\{g_1,g_2,g_3,f\}}$ for *every* quadruple of functions $\{g_1,g_2,g_3,f\}$. The constraint $C'_{j,\{g_1,g_2,g_3,f\}}$ is the conjunction of five constraints, defined below.

The first two constraints ensure that the variables $y_{j,*}$ define proper functions:

1. $y_{j,g_1} + y_{j,g_2} = y_{j,g_1+g_2}$
2. $y_{j,g_1} \cdot y_{j,g_2} = y_{j,g_1g_2+g_3} - y_{j,g_3}$

The first test evidently ensures that function evaluation commutes with addition. The second test does the same for multiplication, with the slight caveat of adding and subtracting the function $g_3$. Note that if the test omitted $g_3$ then both sides of the constraint would be 0 with probability $\frac{3}{4}$, which would make the distribution of the queries $y_{j,g_1g_2}$ significantly non-uniform, and would let us "cheat" when assigning variables to $y_{j,*}$.

The third test will ensure that the pair $(a,b)$ represented by $y_{j,*}$ actually satisfies $P$. One deterministic way to do this is to note that $P$ is a function from $\Gamma \times \Gamma$ into $\{0,1\}$, so we can evaluate $y_{j,P}$ and constrain it to be 1. But as above, this test would not be robust enough since it samples $y_{j,*}$ in a highly non-uniform manner. We modify it to the following:

3. $y_{j,P+g_3} - y_{j,g_3} = 1$

We now note that even if we have succeeded so far in transforming each constraint $C_j = P(x_{j_1}, x_{j_2})$ into a way to express the variables $x_{j_1}, x_{j_2}$, we must still ensure that when the same variable $x_i$ appears in multiple constraints these representations correspond to a consistent value in $\Gamma$. We do this by relating the variables $y_{j,*}$ to the variables $x'_{j_1,*}$ and $x'_{j_2,*}$.

Given a univariate function $f$, let $f^{(1)}$ denote the bivariate function $f^{(1)}(a,b) \overset{\text{def}}{=} f(a)$ that uses only its first input. Define $f^{(2)}$ similarly to be the function that uses only its second input. We create the following two constraints to ensure consistency of all the representations of the original variables:

4. $y_{j,f^{(1)}+g_3} - y_{j,g_3} = x'_{j_1,f}$
5. $y_{j,f^{(2)}+g_3} - y_{j,g_3} = x'_{j_2,f}$

From here, Dinur shows that if at least some fixed fraction of the constraints $C_{j,*}$ are satisfied then the involved variables can be "decoded" to a satisfying assignment of the original problem. If this fraction were (say) 99%, then this would imply that the unsatisfiability gap decreases by at most a factor of 100 under this transformation. We omit the details and move on to a proof of Lemma 1.

## 4 Proof of Lemma 1

We show how to amplify the unsatisfiability gap of a max-2-SAT-$\Sigma$ problem. We motivate our approach with some natural ideas that do not quite work.

The most natural approach is to replace constraints of the original problem with *pairs* of constraints. Thus for every pair $C_{j_1}, C_{j_2}$ we would form a new constraint that consists of the conjunction $C_{j_1} \wedge C_{j_2}$. If at most $1-\epsilon$ fraction of clauses were satisfied in the original problem, then at most $(1-\epsilon)^2$ fraction will be satisfied in the new problem, which will essentially double the unsatisfiability gap.

However, note that after the above transformation, constraints now operate on *four* variables at a time, instead of the required two. A second idea is to now replace variables with pairs of variables, so that each constraint will now operate on a pair of pairs of variables, instead four individual variables. In this manner, we could transform max-2-SAT-$\Sigma$ problems into max-2-SAT-$\Sigma^2$ problems, but we might no longer be sure about the unsatisfiability gap. It turns out there is a theorem called the "Parallel Repetition Theorem" that would imply the desired result if instead of taking pairs of everything, we took $l$-tuples, for large enough $l$. This approach, however, introduces nonlinear blowup to the number of constraints and variables, which we cannot afford.

The problem here is that we are trying to sample random $l$-tuples by using $l$ times as many (random) index bits, and we cannot afford linear blowup in the number of index bits. The idea is to use some kind of pseudo-random generator to get the effect of $l$-tuples using a generator that uses only *additively* more bits.

### 4.1 Amplification of BPP

In this section we recall a standard result on how to amplify the success probability of BPP algorithms using "recycled" randomness.

Let $L$ be some language in BPP. By definition, there exists a Turing machine $A$ using $r(n) = \text{poly}(n)$ random bits such that

$$\forall x \in \{0,1\}^n, \Pr_{R \leftarrow \{0,1\}^{r(n)}}[A(x,R) = L(x)] \geq \frac{2}{3}.$$

We note that using more random bits, we can amplify this probability. Specifically, using $q(n)r(n)$ random bits, for some polynomial $q$, we can increase the success rate from $\frac{2}{3}$ to $1-2^{-q(n)}$ by repeating the above experiment $q$ times and taking the majority outcome.

The drawback here, is that we require $q$ times more randomness.

A natural idea to try is to generate *pairwise* random numbers. We can generate pairwise independent numbers from a seed consisting of just a pair of random numbers, and thus use only $2r(n)$ random bits. However, because the resulting pseudo-randomly generated numbers are not independent, the success rate does not amplify as above. Specifically, if for fixed $x$ we consider $\{0,1\}^{r(n)}$ as being divided into two sets, on one of which $A$ returns the right answer, on one of which $A$ returns the wrong answer, we want a guarantee that not too many of the pseudo-random samples will lie in the "wrong" set. We would like to apply something like the Chernoff bound, but unfortunately that assumes complete independence.

## 4.2 Expander Walks

A more hopeful idea is to construct a sequence of numbers in $\{0,1\}^{r(n)}$ by taking a random walk on a graph with vertex set $\{0,1\}^{r(n)}$. If the degree of our graph is small, some constant $d$, then each new sample requires only $\log d$ more bits to specify the next leg of a random walk.

Explicitly, given a graph $G$, a random walk on $G$ is a sequence $R_j$ constructed as follows.

- Pick $R \in G$ at random.
- Pick $q(n)$ choices $i_1, ..., i_{q(n)} \in \{1, ..., d\}$ at random.
- Let $R_1 = R$, and iteratively define $R_j$ to be the $i_j$th vertex adjacent to $R_{j-1}$.

We ask here whether such a random walk will necessarily produce elements of the right independence properties. We immediately note that if the graph $G$ is composed of two disjoint pieces, then the choice of the starting point will determine which half the rest of the walk falls into, and that such such walks will have very bad properties for us. We want to ensure that such a case does not happen, or in other words, that the graph is very well connected.

It turns out that expander graphs provide the desired properties.

Recall that a graph $G = (V, E)$ is called an $\epsilon$-expander if for all sets of vertices $S \subset V$ that are sufficiently small, $|S| \leq \frac{|V|}{2}$, the number of edges crossing from $S$ to its complement is large:

$$|E(S, \overline{S})| \geq \epsilon |S|.$$

We claim the following:

**Claim 6** *If $G$ is an $\epsilon$-expander then a random walk on $G$ hits every set $S$ roughly $\frac{|S|}{|V|}$ fraction of the time, where "roughly" is interpreted to correspond to some kind of Chernoff bound.*

This means that with $r(n) + d \cdot q(n))$ randomness, we can increase our probability of success in an exponential manner by applying these Chernoff-type bounds.

## 5 Dinur's Amplification

We now briefly outline how this technique may be applied to the max-SAT problem.

Given a max-2-SAT problem $\phi$ with variables $x_1, ..., x_n$ and constraints $C_1, ..., C_m$, define the *constraint graph* $G$ of $\phi$ to be the graph on $n$ vertices and $m$ edges such that each vertex corresponds to a variable $x_i$, and each constraint becomes an edge that connects the two variables involved in the constraint.

We then "convert" $G$ into a $d$-regular $\epsilon$-expander graph. The construction proceeds in two steps: the first step is to reduce the degree of $G$ to a constant $d$ without changing $G$ "too much" (this is a zig-zag product); the second step is to add edges until the graph has good expansion properties. Denote this version of $G$ by $G'$.

Our plan is now to take a random walk on $G'$, and verify the constraints corresponding to every edge we cross. Because of the above claim about expanders, such a choice of constraints will be suitably random.

Recall however, that since many vertices of $G'$ may correspond to the same vertex of $G$, we have to add a compatibility check to ensure that a variable $x_i$ takes consistent values in its different incarnations.

To do this, we apply the familiar trick of replacing $G'$ with some *much* huger graph $G''$ that contains enough redundancy to allow for probabilistic verification.

Specifically if $t$ is the length of our intended random walk, let $G''$ be the multi-graph whose vertices are the same as those in $G'$, but which has an edge for every *path* of length at most $t/2$ in $G'$. Furthermore, we append to the label of each vertex of $G''$ the label in $G'$ of each of its $\leq d^{t/2}$ neighbors. We note that $G''$ has alphabet $\Sigma^{d^{t/2}}$.

The constraint associated with an edge $(u, v)$ of $G''$ is satisfied if all of the assignments in $G'$ at the intersection of the radius $t/2$ neighborhoods of $u$ and $v$ are consistent, and satisfy all their constraints from $G'$.

Note that while such checks involve a huge amount of data, all this data is contained in the labels of the vertices $u, v \in G''$, so this is still an instance of max-2-SAT.

We omit the various robustness proofs for this procedure, but note the following "decoding" algorithm that will help show that anything almost-satisfying $G''$ can be decoded to an assignment that almost-satisfies $G$.

> **Decoding assignments to $G''$:** Given an assignment to vertices of $G''$, to find a corresponding assignment to a vertex $u \in G'$, take a random walk of length $\leq \frac{t}{2}$ from $u$, and whichever vertex $v$ we end up at, find its corresponding vertex in $G''$, and inspect its label and return its opinion of the value of $u$.

In the next lecture we will consider "recycled randomness" in more depth.