The PCP Theorem and Hardness of Approximation

Naman Agarwal, Subhro Roy, Ali Vakilian

December 16, 2012

Abstract

In this report, we give a brief overview of Probabilistically Chekable proofs, and present Irit Dinur’s proof of the celebrated PCP theorem. We also briefly deal with the importance of the PCP theorem in deciding hardness of approximation of various problems.

1 What are PCPs?

Let us recap what we mean by \( \text{NP} \). \( \text{NP} \) is the class of all decision problems which have deterministic polynomial time verifiers. The verifier will have access to the input \( x \) and a proof \( \pi \) given by an all-powerful prover, and it can check in polynomial time whether \( \pi \) is a valid proof determining \( x \)’s membership in \( L \). So, if \( x \) indeed belongs to \( L \), there will exist a proof \( \pi \), which will make the verifier accept. Otherwise, no proof can make the verifier accept. Formally, a language \( L \in \text{NP} \) has a polynomial time verifier \( V \) such that the following holds

- (Completeness) If \( x \in L \), there exists a proof \( \pi \), such that \( V^\pi(x) \) accepts. (\( V^\pi(x) \) refers to the output of \( V \) on input \( x \) and proof \( \pi \))
- (Soundness) If \( x \notin L \), for any proof \( \pi \), \( V^\pi(x) \) rejects.

Consider the example of graph 3-colorability problem. We know that this problem is \( \text{NP} \)-Complete. The input to the problem is a graph \( G \), and we want to determine whether \( G \) can be colored by at most 3 colors. The proof \( \pi \) for the input \( G \) is actually an assignment of colors to the vertices of \( G \). Now how will the verifier work? It will go to every edge of the graph \( G \), and check whether the ends of it are colored with different colors. Even if one of the edges does not satisfy the property, the verifier will reject.

Now, what if we want a probabilistic verifier \( V \) for the above problem. We allow some randomness to the verifier, but at the same time, allow it to only see a constant number of positions of the proof. We want the property that whenever a particular input \( x \) belongs to \( L \), the verifier always accepts, and when \( x \) does not belong to \( L \), the verifier rejects with probability greater than \( 1/2 \). In order to have the aforementioned property, intuitively it seems that whenever some \( x \) does not belong to \( L \), the proof has a lot of errors in it. In that case, even when the verifier checks a few bits, there is a high probability that it will detect the error, if it is an erroneous proof.

Consider our example of the 3-coloring problem. A wrong proof might give a wrong color only to a single edge, and hence the proof will not be probabilistically checkable by probing only a constant number of bits. However, can we rewrite the proof in a different form, such that even a small error is expanded and distributed throughout the proof? The answer is a big resounding YES. It is due to the following theorem.

Theorem 1.1. (PCP Theorem) All languages \( L \subseteq \text{NP} \) have a PCP system wherein on input \( x \in \{0, 1\}^n \):

- Prover \( P \) writes down a \( \text{poly}(n) \) length proof.
• Verifier $V$ looks at $x$ and does polynomial-time deterministic computation. Then $V$ uses $O(\log n)$ bits of randomness to choose $C$ random locations in the proof. Here $C$ is a absolute universal constant, say 100. $V$ reads the bits in the $C$ randomly chosen locations from the proof, and does some tests on them, accepting or rejecting.

• Completeness: If $x \in L$ then $P$ can write a proof that $V$ accepts with probability 1.

• Soundness: For every $x \notin L$, no matter what proof $P$ writes, $V$ accepts with probability at most 1/2.

In order to prove the theorem, we have to show for any \textbf{NP} language, we can have a probabilistically checkable proof. As in the \textbf{NP}-Complete reductions, if we show that some \textbf{NP}-Complete problem has Probabilistically Checkable proof, we are done. For any \textbf{NP} problem, we can just reduce it to the \textbf{NP}-Complete problem, and then use the proof for the \textbf{NP}-Complete problem. The \textbf{NP}-Complete problem we choose here is the Graph 3-Coloring Problem.

1.1 Hardness of Approximation

The PCP Theorem directly implies hardness of approximation results. To explain that, we fist explain the notion of a constraint satisfaction problem.

\textbf{Definition 1.2.} Let $V = \{v_1, \ldots, v_n\}$ be a set of variables, and let $\Sigma$ be a finite alphabet. A $q$-ary constraint $(C, i_1, \ldots, i_q)$ consists of a $q$-tuple of indices $i_1, \ldots, i_q \in [n]$ and a subset $C \subseteq \Sigma^q$ of acceptable values. A constraint is satisfied by a given assignment $a : V \rightarrow \Sigma$ iff $(a(v_{i_1}), a(v_{i_2}), \ldots, a(v_{i_q})) \in C$. The constraint satisfaction problem is the problem of, given a system of contraints $C = \{c_1, \ldots, c_n\}$ over a set of variables $V$, deciding whether there is an assignment for the variables that satisfies every constraint.

Many \textbf{NP}-Complete problems are special cases of the \textbf{CSP}, so it is obviously \textbf{NP}-Complete. For example, in the equivalent of the 3-colorability problem, the alphabet is $\Sigma = \{1, 2, 3\}$ and the binary constraints are of the form $(C, i_1, i_2)$ where $C = \{(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)\}$ consists of 6 out of the possible 9 values that exclude equality.

An optimization version of this problem, called max-CSP, is to find an assignment that satisfies a maximum number of constraints. Let us denote unsat-value of $C$, denoted $\text{UNSAT}(C)$, defined to be the smallest fraction of unsatisfied constraints, over all possible assignments for $V$. Clearly $C$ is satisfiable if and only if $\text{UNSAT}(C) = 0$. In this notation, the following theorem is an equivalent form of the PCP theorem.

\textbf{Theorem 1.3.} (Inapproximability version of the PCP theorem) There are integers $q > 1$ and $|\Sigma| > 1$ such that, given as input a collection $C$ of $q$-ary constraints over an alphabet $\Sigma$, it is \textbf{NP}-hard to decide whether $\text{UNSAT}(C) = 0$ or $\text{UNSAT}(C) \geq 1/2$.

\textbf{Lemma 1.4.} \textit{Theorem 9.1 and Theorem 1.3 are equivalent}

\textbf{Proof:}

• ($\Rightarrow$) According to \textit{Theorem 9.1}, every \textbf{NP} language has a verification procedure $V$ that reads $c \log n$ random bits, accesses $q = O(1)$ bits from the proof and decides whether to accept or reject (where $q$ and $c$ are constants). For each fixed random bit pattern $r \in 0, 1^{c \log n}$, $V$ (deterministically) reads a fixed set of $q$ bits from the proof: $i_1^{(r)}, \ldots, i_q^{(r)}$. There is a set of possible contents of these bits (say $C^{(r)}$), which will make the verifier $V$ accept. We want to code this as a boolean formula.

Let $\sigma = \{0, 1\}$ and we put a Boolean variable for each proof location accessed by $V$ on input $x$ (so there are at most $q^{2^{c \log n}} = qrc$ Boolean variables). Next, we construct a system of constraints, with the conditions $C^{(r)}$, those conditions which will make the verifier accept.
It remains to observe that the rejection probability of $V$ is exactly equal $\text{UNSAT}(C_x)$ so it is zero if $x \in L$. When $x \notin L$, we have that for every random string $r$, there is at least one constraint that cannot be satisfied. Therefore, we have that a constant fraction of the constraints cannot be satisfied. So, $\text{UNSAT}(C_x) = \text{Constant}$, which can be made greater than $1/2$ by repeating the check independently in parallel.

- $(\Leftarrow)$ For the converse, assume there is a reduction taking instances of any NP-language into constraint systems such that the gap property holds. Here is how to construct a verifier. The verifier will first (deterministically) compute the constraint system output by the reduction guaranteed above. It will expect the proof to consist of an assignment for the variables of the constraint system. Next, the verifier will use its random string to select a constraint uniformly at random, and check that the assignment satisfies it, by querying the proof at the appropriate locations.

$\square$

Dinur proves the inapproximability version of the PCP theorem. The deduction starts from the NP-hard problem GRAPH 3-COLORABILITY. This problem is reduced to a particular gap problem called $\text{GAP-CG}_{1,s}$.

**Definition 1.5.** A constraint graph is given by an alphabet $\Sigma$, a graph $G=(V,E)$ and a set of contraints $C = \{c_e \subseteq \Sigma \times \Sigma \mid e \in E\}$. A labelling on $G$ is an assignment $\sigma : V \rightarrow E$ of elements from $\Sigma$ to each vertex of $G$. A labelling $\sigma$ satisfies an edge $(u,v) \in E$ if $(\sigma(u), \sigma(v)) \in E$. The gap problem for constraint graphs $\text{GAP-CG}_{1,s}$ is to distinguish between the cases (1) there is a labelling which satisfies all the edges and (2) every labelling fails to satisfy at least a fraction $s$ of edges.

Note that the graph 3-coloring problem is a special case of constraint graph satisfiability problem. The way Dinur’s proof works is to start with the fact that 3-COLOR is NP-hard, from which one immediately deduces that $\text{GAP-CG}_{1,1-1/m}$ is NP-hard, where $m$ is the number of edges. (This is because in any illegal 3-coloring, at least one edge must be violated.) The proof will try to amplify the soundness gap from $1/m$ up to some universal constant. Since constraint graph satisfiability is also a particular type of CSP, showing this reduction is sufficient for proving the PCP theorem, by Theorem 1.3.

At each stage the proof will be working with a constraint graph $G$. In the most important step of the proof, a new graph $G^t$ is constructed from $G$, where the constraints in $G^t$ correspond to walks in $G$ of length $t$. If the constraint graphs are nicely structured (i.e., are constant-degree expanders) then these walks in $G$ will mix nicely.

## 2 Overview

We know that $\text{GAP-CG}_{1,1-1/m}$ is NP-hard. Our goal is to make a polynomial-time reduction from this problem graph $G_{\text{init}}$ to a graph $G_{\text{final}}$, such that, if $G_{\text{init}}$ is satisfiable, so is $G_{\text{final}}$ and if $G_{\text{init}}$ is not satisfiable (at least $1/m$ fraction of edges not satisfiable), at least $s$ fraction of constraints of $G_{\text{final}}$ are not satisfied.

We will design four polylime subroutines, which when applied to a constraint graph $G$, gives another constrained graph $G'$, such the gap of $G'$ is twice the gap of $G$, whereas other parameters are not hampered much. Repeating this transformation $O(\log n)$ times will gives a constant gap of $s$. Here we lay down the four transformations: (Input graph $G$, output graph $G'$)

- **Degree Reduction** We strat with the constant sized alphabet, $\Sigma_0$. Goal of this phase is to construct regular $G'$ with $\text{deg}' = d_0$, where $\text{deg}'$ is a universal constant. At the end of the phase:
  - $\text{size}' = O(\text{size})$ (i.e., the size goes up by a constant factor).
The alphabet does not change.
- If \( \text{gap} = 0 \) then \( \text{gap}' = 0 \).
- Otherwise, \( \text{gap}' \geq \text{gap}/O(1) \).

**Expanderization** We continue with the output of the previous phase. Given a regular graph \( G \) with constant degree \( \text{deg} \), the goal is to construct a constant degree expander \( G' \), i.e., \( G' \) is \( d_1 \)-regular for some universal constant \( d_1 \), and \( \lambda' < d_1 \). Similar to Degree Reduction phase at the end of the phase, the size of graph grows up by a constant and gap value diminishes by a constant value.

**Gap amplification** This is the critical step and the main novelty in Dinur’s proof. Assumption: \( G \) is an \((n,d,\lambda)\)-expander, with \( \lambda < d \) universal constants; and, the alphabet is \( \Sigma_0 \), a constant. This phase, improves the unsatisfiable gap by a factor of \( \sqrt{t} \), where \( t \) is a fixed parameter. Specifically:

\[
\text{gap}' \begin{cases} 
0 & \text{if } \text{gap} = 0 \\
\geq O(\sqrt{t}) \cdot \min(\text{gap}, 1/t) & \text{else}
\end{cases}
\]

In other words, if \( G \) was satisfiable then so is \( G' \); however, if \( G \) was not satisfiable, then \( G' \) has gap which is larger by a factor of \( \sqrt{t} \), unless this is already bigger than the universal constant \( 1/t \), in which case it just becomes at least \( 1/t \). At the end of this phase,

- \( \text{size}' = O(\text{size}) \).
- The new alphabet \( \Sigma \) is a much huger constant; something like \( \Sigma_0^d \).
- \( \text{deg}', \lambda' \) may become bad as well.

**Alphabet Reduction** In this phase, alphabet size goes down to \( \Sigma' = \Sigma_0 \). However,

- \( \text{size}' = O(\text{size}) \), where the constant depends on the input alphabet size \( |\Sigma| \). (The dependence is very bad, in fact; at least exponential.)
- If \( \text{gap} = 0 \) then \( \text{gap}' = 0 \).
- Otherwise, \( \text{gap}' \geq \text{gap}/O(1) \).

## 3 Degree Reduction

We assume that that there exists constants \( \lambda_0 < d_0 \) such that \((n,d_0,\lambda_0)\)-expanders can be explicitly constructed in polynomial time. Given an input constrained graph \((G,C)\), we apply the following transformation:

- Replace each vertex \( u \in V \) by \( \text{deg}(u) \) many vertices to get the new vertex set \( V' \). Denote the set of new vertices corresponding to \( u \) by \( \text{cloud}(u) \). Each vertex in \( \text{cloud}(u) \) naturally corresponds with a neighbor of \( u \) from \( G \).
- For each edge \((u,v) \in E\), place an inter-cloud edge in \( E' \) between the associated cloud vertices. This gives exactly one inter-cloud edge per vertex in \( V' \). Whatever the old constraint on \((u,v)\) was, put the exact same constraint on this inter-cloud edge.
- For each \( u \in V \), put a \((\text{deg}(u),d_0,\lambda_0)\)-expander on \( \text{cloud}(u) \). Further, put equality constraints on these expander edges.

We note that each new vertex gets only one edge in the second step, and \( d_0 \) edges in the third step. Hence we have formed a \((d_0 + 1)\)-regular graph. The number of vertices are \( \sum_{u \in V} \text{deg}(u) = |E| \), and hence number of edges is \( |E|(d_0 + 1) \). Therefore, the overall size goes up only by a constant factor. Also, if the old graph was satisfiable, the new graph will also be satisfiable. We only need to show that in the NO case, \( \text{gap}' \geq \text{gap}/O(1) \).
Lemma 3.1. \( \text{gap}' \geq \text{gap}/O(1) \)

**Proof:** Let \( \sigma' : V' \rightarrow \sum_0 \) be a best assignment for \((G', C')\). To relate the fraction of edges in \( E' \) that \( \sigma' \) satisfies back to the gap in the old constraint graph, we define an assignment \( \sigma : V \rightarrow \sum \) as follows: \( \sigma(u) \) is defined to be the plurality vote of \( \sigma' \) on \textit{cloud}(u). We know that \( \sigma \) violates at least \( \gamma |E| \) constraints in the old graph, where \( \gamma \) is the gap.

We define \( S^u \) to be the set of vertices in \textit{cloud}(u) on which \( \sigma' \) disagrees with \( \sigma(u) \). Suppose \( e = (u,v) \) is one of the edges which have been labelled wrongly (at least \( \gamma |E| \)) by \( \sigma \). Let \( e' \) be the corresponding inter-cloud edge in \( E' \). The key observation is the following: Either \( \sigma' \) violates the edge \( e' \) or one of the endpoints of \( e' \) belongs to \( S^u \) or \( S^v \). Thus we conclude:

\[
\gamma |E| \leq (\text{# edges violated by } \sigma') + \sum_{u \in V} |S^u|
\]

We have two cases now:

**Case 1:** \# edges violated by \( \sigma' \) is at least \( \gamma/2 \). Since \( \sigma' \) is the best assignment for \((G', C')\), we get

\[
\text{gap}' = \text{# edges violated by } \sigma' \geq \gamma/2 \cdot |E'| = \gamma \cdot 2(d_0 + 1)|E'|
\]

Hence, \( \text{gap}' \geq \text{gap}/O(1) \).

**Case 2:** \( \sum_{u \in V} |S^u| \geq \gamma/2 \cdot |E'| \)

For any \( u \in V \) let \( S^u_a \) denote the vertices in \( S^u \) which \( \sigma' \) labels by \( a \in \Sigma_0 \). By definition of \( S_u \) as the non-plurality set, we must surely have \( |S^u_a|/|\text{cloud}(u)| \leq 1/2 \). Thus by the fact that the cloud is an expander, we get that there are at least \( \Omega(1) \cdot |S^u_a| \) edges within the cloud that come out of \( S^u_a \). Every such edge is violated by \( \sigma' \), since these edges have the equality constraint. Thus the number of edges that \( \sigma' \) violates is at least

\[
\sum_{u \in V} \sum_a \Omega(1)|S^u_a| = (1/O(1)) \sum_{u \in V} |S^u| \geq (1/O(1))(\gamma/2)|E| = (1/O(1))(\gamma/2)(|E'|/(d_0 + 1)) = (\gamma/O(1))|E'|
\]

Thus we have the desired property. \( \square \)

4 **Expanderizer**

Given a constraint graph \( G \) with each vertex having degree \( d \), we wish to get a constraint graph \( G' \) which is also a regular constraint graph, and is also a good expander. All we need to do is superimpose on \( G \), an \((n, d_0, \lambda_0)\)-expander. On each edge of the expander, we put “null” constraint, that is, a constraint which is always satisfied.

This new graph \( G' \) has a degree \( d + d_0 \), a constant. The number of edges is \( n(d + d_0)/2 \). Regarding the properties of the gap, if the original gap was 0, the new gap is still 0. Consider an assignment \( \sigma' \) for the new constraint graph. Viewing it as an assignment for the old constraint graph, we have that it must violate at least
gap|E| old constraints. Since these constraints are the same in the new graph as well, we know that at least gap|E| constraints are violated in the new graph as well. The number of edges of the new graph is O(|E|). Therefore, every assignment in the new graph violates at least gap/O(1) fraction of the constraints. What remains to be shown, is that, this new graph is also a constant degree expander. It is a direct consequence of the following lemma.

**Lemma 4.1.** If \( G \) is a \( d \)-regular graph on the vertex set \( V \) and \( H \) is a \( d' \)-regular graph on \( V \) then \( G' = G \cup H = (V, E(G) \cup E(H)) \) is a \((d + d')\)-regular graph such that

\[ \lambda(G') \leq \lambda(G) + \lambda(H) \]

**Proof:** Choose \( x \) such that \(|x| = 1\), \( x \rightarrow \Gamma = 0 \), and \( \lambda(G') = < A^{G'} x, x > \). Now,

\[ < A^{G'} x, x >= < A^{G} x, x > + < A^{H} x, x > \]

(since \( G' = G + H \))

\[ \leq \lambda(G) + \lambda(H) \]

(since \( \lambda(G') = \max_{x \in R^n, x, \Gamma = 0, x \neq 0} \frac{| < A^{G'} x, x > |}{< x, x >} \) \)

\( \square \)

5 Gap Amplification

As discussed before the key idea in this approach is to amplify the number of unsat value. To amplify the unsat value of a constraint graph we raise it to power \( t \), for some constant \( t \). In the following we describe the operation: Let \( G = (V, E), \Sigma, C \) be a \( d \)-regular constraint graph, and let \( t \in \mathbb{N} \). A sequence of \((u_0, \cdots u_t)\) is called a \( t \)-step walk in \( G \) if for all \( 0 \leq i \leq t \), \((u_i, u_{i+1}) \in E \). We define \( G^t = (V, E^t), \Sigma^{d^{[t/2]}}, C^t \) to be the following constraint graph:

- The vertices of \( G^t \) are the same as the vertices of \( G \).
- Edges: \( u \) and \( v \) are connected by \( k \) parallel edges in \( E^t \) if the number of \( t \)-step walk from \( u \) to \( v \) in \( G \) is exactly \( k \).
- Alphabet: The alphabet of \( G^t \) is \( \Sigma^{d^{[t/2]}}, \). Let \( \Gamma(u) = \{u' \in V \mid (u = u_0, \cdots, u_{[t/2]} = u' \text{ is a walk in } G)\} \). Clearly \(|\Gamma(u)| \leq \Sigma^{d^{[t/2]}}, \). We interpret a value \( a \in \Sigma^{d^{[t/2]}}, \) as an assignment \( a : \Gamma(u) \rightarrow \Sigma \). Thus value of each vertex \( u \) describe \( u \)'s opinion of its neighbors’ values too.
- Constraint: The constraint associated with an edge \( e = (u, v) \in E^t \) is satisfied by a pair of values \( a, b \in \Sigma^{d^{[t/2]}}, \) iff there exists a function \( \sigma : \Gamma(u) \cup \Gamma(v) \rightarrow \Sigma \) that satisfies every constraint \( c(e) \in E \cap \Gamma(u) \times \Gamma(v) \) and

\[ \forall u' \in \Gamma(u), v' \in \Gamma(v), \sigma(u') = a(u') \text{ and } \sigma(v') = b(v') \]

where \( a(u') \) is the value \( a \) assigns \( u' \) and \( b(v') \) is the value \( b \) assigns \( v' \).

Notice that by this definition of \( C^t \), since the constraint on the edge \( u, v \) depends only on \( u \) and \( v \) and not the path between them that the edge represent, every two parallel edges enforce the same constraint. However, we consider them because they enhance the percentage of the unsatisfied constraints when they are violated. If \( \text{UNSAT}(G) = 0 \) then clearly by using the same assignment of \( G \), \( \text{UNSAT}(G^t) = 0 \). More important observation is that if \( \text{UNSAT}(G) \geq 0 \) then the unsat value if multiplied by a factor of \( \sqrt{t} \). It requires the underlying graph be \( d \)-regular for constant \( d \) and has self-loops.
Figure 1: An example of imposed constraint in $G^t$.

**Lemma 5.1. (Amplification Lemma)** Let $0 < \lambda < d$, and $|\Sigma|$ be constants. There exists a constant $\beta_2 = \beta_2(\lambda, d, |\Sigma|) > 0$, such that for every $t \in \mathbb{N}$ and for every $d$-regular constraint graph $G = \langle (V, E), \Sigma, (C) \rangle$ with a self-loop on each vertex and $\lambda(G) \leq \lambda$, 

$$\text{UNSAT}(G^t) \geq \beta_2 \sqrt{t} \cdot \min(\text{UNSAT}(G), 1/t).$$

This lemma proves that we can amplify the unsat value by a factor of $\sqrt{t}$ while the size of the constraint graph increase linearly. This lemma is tricky since it might be the case that an assignment to $G^t$ does not correspond to any obvious assignment for $G$. In other words for a vertex $w \in V(G)$ and pair of vertices $(u, v)$ in $G^t$ that contain $v$ in their $t/2$-neighborhood, it is possible that $a(v)_w = \neq b(u)_w$ where $a$ is the assignment of $v$ and $b$ is the assignment of $u$. Thus we need to work with a certain assignment for $G$ obtaining from $G^t$ as follows:

The popular opinion or the plurality assignment: For every $\vec{\sigma} : V \rightarrow \Sigma^{|t/2|}$ let $\sigma : V \rightarrow \Sigma$ be defined such that for each $v \in V$,

$$\sigma(v) := \max_{a \in \Sigma} \{Pr[A \text{ random } \lceil t/2 \rceil \text{-step walk from } v \text{ reaches a vertex } w \text{ for which } \vec{\sigma}(w)_v = a]\},$$

where $\bar{\sigma}(w)_v \in \Sigma$ denotes the restriction of $\bar{\sigma}(w)$ to $v$. This value represent the most likely value of $v$ in its neighbor opinion.

In order to prove Amplification Lemma, we prove the following stronger lemma which is based on “the popular opinion” setting.

**Lemma 5.2.** Let $0 < \lambda < d$, and $|\Sigma|$ be constants. There exists a constant $\beta_2 = \beta_2(\lambda, d, |\Sigma|) > 0$, such that for every $t \in \mathbb{N}$ and for every $d$-regular constraint graph $G = \langle (V, E), \Sigma, (C) \rangle$ with a self-loop on each vertex and $\lambda(G) \leq \lambda$ the following holds. For every $\vec{\sigma} : V \rightarrow \Sigma^{d^{t/2}}$ let $\sigma : V \rightarrow \Sigma$ be defined according to “popular opinion”. Then,

$$\text{UNSAT}_\sigma(G^t) \geq \beta_2 \sqrt{t} \cdot \min(\text{UNSAT}_\sigma(G), 1/t).$$

First, let give some intuition about Lemma 5.2 and why it holds. One simple way to amplify the probability of finding a falsified edge in a query is to consider $t$ random edges instead of one. The probability that an assignment falsify at least one the edges is $t$ times the chance of picking a falsified edge. Moreover, since $G$ is an expander, the probability remains roughly the same even if the edges are chosen by taking a random length-$t$ walk in $G$.

The graph $G^t$ is constructed in a way to reflect this behavior. In fact, for the case that $\vec{\sigma}(\sigma) : V \rightarrow \Sigma^{d^{t/2}}$ is loyal to an underlying assignment $\sigma : V \rightarrow \Sigma$ (which means $\vec{\sigma}(v)_w = \sigma(w)$ for all $w$ reachable from $v$ by a
walk of length $t/2$) then $\text{UNSAT}(G')$ change as we desired. But, the proof of Lemma 5.2 in general case, since it may be the case that there is no assignment $\sigma$ such that $\tilde{\sigma}(\sigma)$ follows.

The idea is to work with the “popular opinion” assignment $\sigma$. For an assignment $\tilde{\sigma}$, consider the “popular opinion” assignment as described before. By the definition of $\text{UNSAT}$, the fraction of falsified edges in assignment $\sigma$ is at least $\text{UNSAT}(G)$. Now, we need to prove that an edge of $G'$ (random walk of length $t$ in $G$) is rejected with constant probability.

Let $E' = E(G')$ be the set of edges in $G'$. There is a one to one correspondence between edges $e \in E'$ and walks of length $t$ in $G$. By abusing the notation, we write $e = (v_0, \cdots, v_t)$ where $(v_{i-1}, v_i) \in E$ for all $1 \leq i \leq t$.

**Definition 5.3.** A walk $e = (v_0, \cdots, v_t)$ is hit by its $i$-th edge if

- $(v_{i-1}, v_i) \in F$, and
- Both $\tilde{\sigma}(v_0)_{v_{i-1}} = \sigma(v_{i-1})$ and $\tilde{\sigma}(v_i)_{v_i} = \sigma(v_i)$.

**Proof Sketch of Lemma 5.2:** Consider the “popular opinion” assignment $\sigma$ associated with $\tilde{\sigma}$. Let define random variable $X_v$ to be the value of $\tilde{\sigma}(w)_{v}$ for a $w$ in a $t/2$-step far from $v$. By definition of “popular opinion”:

$$Pr[X_v = \sigma(v)] \geq \frac{1}{|\Sigma|}.$$ 

Let $F$ be the subset of falsified edges in $\sigma$ if $\text{UNSAT}_{\sigma}(G) \leq 1/t$. Otherwise, we take an arbitrary subset of edges such that $|F| = \lceil |E|/t \rceil$. Thus,

$$\frac{|F|}{|E|} \leq \min(\text{UNSAT}_{\sigma}(G), 1/t)$$

Let $I = \{t/2 - \sqrt{t}/2 \leq i < t/2 + \sqrt{t}/2\} \subset \mathbb{N}$ be the set of “middle” indices. For each edge $e \in E'$, $N(e)$ is defined as follows:

$$N(e) = |\{i \mid e \text{ is hit by its } i\text{-th edge}\}|.$$ 

Clearly, $N(e)$ is an value between 0 and $\sqrt{2t}$. Since we are interested in $\text{UNSAT}_{\tilde{\sigma}}(G')$ a useful observation is:

$$Pr[N(e) > 0] \leq Pr[e \text{ rejects } \tilde{\sigma}] = \text{UNSAT}(G').$$

By following lemmas (Lemma 5.4 and Lemma 5.5) and Fact 5.1, we will prove that:

$$\Omega(\sqrt{t}). \frac{|F|}{|E|} \leq Pr[N(e) > 0]$$

and this shows that

$$\Omega(\sqrt{t}). \min(\text{UNSAT}_{\sigma}(G), 1/t) \leq \Omega(\sqrt{t}). \frac{|F|}{|E|} \leq Pr[N(e) > 0] \leq \text{UNSAT}_{\tilde{\sigma}}(G').$$

which completes the proof. \hfill \Box

**Lemma 5.4.**

$$\mathbb{E}[N(e)] \geq \Omega(\sqrt{t}). \frac{|F|}{|E|}$$

**Lemma 5.5.**

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

8
Fact 5.1.

$$\mathbb{E}[N(e)] \geq \mathbb{E}^2[N(e)]/\mathbb{E}[(N(e))^2]$$

Applying Fact 5.1 and substituting from Lemma 5.4 and 5.5 we have the following statement which we used in the proof of Lemma 5.2.

$$\Omega(\sqrt{t}), \frac{|E|}{|E'|} \leq Pr[N(e) > 0].$$

Later Jutla[] by a more involved construction proved that one can achieve amplification of factor $t$. It followed be the same idea but the definition of $E'$ is a bit different. The major difference is in the definition “popular opinion”. In this work, the “popular opinion” is defined by a distribution over ball containing neighbors of distance $\lceil t/2 \rceil$ such that the closer ones get larger weight in the distribution.

6 Alphabet Reduction

After powering the constraint graph as described in gap amplification phase, we get a graph whose alphabet is much larger, $\Sigma_d^{t/2}$. Since we need to repeat this process $O(\log n)$ times, after a while the size of alphabet will be not constant anymore. But, one of the condition in Lemma 5.2 is that the alphabet size is constant. Thus, in order to be able to apply gap amplification $O(\log n)$ many times, we need to reduce the alphabet size. In fact we should keep it linear in the size of the initial $\Sigma$.

One may realize that the problem we are trying to solve is itself a PCP. Our goal is to find a reduced proof verifier such that we can check each constraint by a constant number of queries. Are we in a loop? The good point is that now the graph we need to solve PCP for is much smaller; has constant size. It is known that there exists some reduction which introduces large gap but increase the number of constraints exponentially too. Fortunately, it is not a preventing factor here because the size of each constraint in $G^t$ only depends on $\Sigma$ and is a constant.

The solution is to use composition which is an essential part of all PCP construction. It is more meaningful in the proof-verification setting. The key point is that a system of $q$-ary constraint over $\Sigma$ corresponds to a PCP that reads $q$ symbols from $\Sigma$. The goal behind composition, introduced by Arora and Safra is to compose two PCPs to improve the query complexity of the composed PCP.

The basic idea of composition is to use “inner” verifier. It means that verifier instead of reading $q$ symbols from $\Sigma$ can use an inner verifier that is a proof for the fact that original PCP will accept this length $q$ input. Thus we read even a fewer bits. This approach ends up having many additional proof, one per random string of the original verifier. The subtlety is to ensure the consistency between additional proofs. For constraint systems, the “inner” verifier is a reduction that transform a single constraint over a large alphabet into a system of constraints over new small alphabet. This is applied on all constraints in parallel in a consistent way. Furthermore, it should be done in a way that the UNSAT value of new system does not drop by more than a constant factor (to keep the $\sqrt{t}$ improvement we made in gap amplification phase). In the following we state Composition Lemma informally.

**Lemma 6.1.** Assume the existence of an assignment tester $\mathcal{P}$, with constant rejection probability $\varepsilon > 0$, and alphabet $\Sigma_0$, $|\Sigma_0| = O(1)$. There exists $\beta > 0$ that depends only on $\mathcal{P}$, such that given any constraint graph $G = \langle (V, E), \Sigma, \mathcal{C}\rangle$, one can compute, in linear time, the constraint graph $G' = G \circ \mathcal{P}$, such that $size(G') = c(\mathcal{P}, |\Sigma|).size(G)$, and

$$\beta \cdot UNSAT(G) \leq UNSAT(G') \leq UNSAT(G).$$

An assignment tester is a type of PCP which is used in composition. You can find more about it in Appendix.
This figure is taken from [2]. It shows how to map a binary constraint satisfaction instance (you can think of $G^t$) to a $q$-ary constraint satisfaction problem. Each variable $u_i$ is mapped to a block of binary variables that in the correct assignment contains a specific encoding\footnote{Wlash-Hadamard encoding} of this variable. The $\Pi$ variables insures the consistency of assignment.

7 Appendix

7.1 Assignment Tester

Basically an assignment tester is an algorithm that given a boolean circuit $\Phi$, output a constraint graph $G$. This graph contains the input variables of $\Phi$ as some of its vertices, and its unsat value is related to the satisfiability of $\Phi$ as follows. For a boolean circuit $\Phi$ over $n$ variables, $\text{SAT}(\Phi) \subseteq \{0, 1\}^n$ is the set of assignments that satisfy $\Phi$.

**Definition 7.1.** An assignment Tester with alphabet $\Sigma_0$ and rejection probability $\varepsilon > 0$ is an algorithm $\mathcal{P}$ whose input is a circuit $\Phi$ over boolean variables $X$, and whose output is a constraint graph $G = \langle (V, E), \Sigma_0, C \rangle$ such that $V \supseteq X$, and such that the following hold. Let $V' = V/X$, and let $a : X \rightarrow \{0, 1\}$ be an assignment.

- *(Completeness)* If $a \in \text{SAT}(\Phi)$, there exists $b : V' \rightarrow \Sigma_0$ such that $\text{UNSAT}_{\alpha \cup b}(G) = 0$.

- *(Soundness)* If $s \notin \text{SAT}(\Phi)$ then for all $b : V' \rightarrow \Sigma_0$, $\text{UNSAT}_{\alpha \cup b}(G) \geq \varepsilon.rdist(a, \text{SAT}(\Phi))$.

There is no requirement on the complexity of $\mathcal{P}$. 
8 Introduction to Hardness of Approximation

This short survey paper is a small attempt at describing the far reaching impacts of the idea of probabilistic checking of proofs on the hardness of approximating known hard problems. Since the development of the theory of NP-hardness/completeness, we have had a very strong notion of what problems are believed to be solvable(efficiently) and what problems are not. The idea of reduction and NP-completeness have also given this very strong fundamental property of one basic abstract(rather philosophical) problem as the key to all problems that have this flavour. This basic problem is the idea of how hard is verification with respect to actual solution finding. In other words, is finding solutions as hard as verifying a solution when it is given. If one also thinks of a computational model where in we are allowed to make some non-deterministic guesses, one can see that that finding solutions is simply taking a guess of all solutions(non-deterministically) and verifying the guess. Thus the above dichotomy between verification and solution finding incarnates itself equivalently into the form of whether adding non-determinism to a computational model gives it extra power. We know that this is not true for computational models like the finite automata, but this property for certain other computational models like the turing machines gives rise to the most celebrated open problem in computer science better known as the $P \neq NP$ conundrum. (The reader is referred to any elementary course/book in complexity theory to get a better feel of what was said above).

It is now largely believed that $P \neq NP$(and so does the author), but what can we make of it ? One natural question to ask is if we cannot find exact solutions efficiently can we approximate it with a certain guarantee ? How close with our approximations can we get or how far we will provably be ? These are some of the notions that the following report hopes to address. The general format of the results proved would be of the form that

Definition 8.1. General format of hardness of approximation results For any problem $I$ (there exists or for all) $\epsilon$, such that approximating $I$ to $\epsilon$ ‘closeness’ is hard, unless $P = NP$

Given our general belief about $P \neq NP$ proving results of the above format and also finding the approximation that achieves the optimum ’closeness’ provides a sense of completeness to the theory and certainly seems an exercise which is enriching and interesting barring the other obvious gains.

The following report first formalises some of the notions that author has handwaved about above, moving onto show some concrete hardness of approximation results for MAX-CLIQUE and MAX-LABELCOVER. We further describe Hastad’s PCP which seemingly achieves the best you can do with the optimising the number of queries used and sketch a proof for the major result. This also gives a concrete hardness of approximation result for MAXE3LIN2.

9 The PCP-Theorem and Hardness of Approximation

The notion of probabilistic proofs has been defined earlier in the report. Here we would just give the basic theorems as a recap.

Theorem 9.1. (PCP Theorem) $NP = PCP_{1,1/2}[O(log(n)), O(1)]$

We now define the notion of GAP Problems. In essence these are instances where there is a big gap between two types of solutions that can be present in the system.

Definition 9.2. (GAP Problems) The $GAP – I_{c,s}$ version of any maximization problem $I$ is defined as: Given the parameters of $I$ the output should be as follows:

- YES if $OPT \geq cm$
• NO if \( OPT \leq sm \)
• anything otherwise

The parameter 'n' above is some related parameter to the specific problem, (eg. the number of clauses in 3SAT or the number of edges in 3COLOR etc.). The basic idea is to be able to differentiate between a highly 'satisfying' instance and low 'satisfying instance'.

The big observation that we want to make right now is that if we prove that a particular gap problem with a gap of \([c, s]\) is NP-Hard, what we have essentially proven is that the related optimization is NP-Hard to be approximated to a factor of \(s/c\). To see why this is true we can imagine using the \( s/c \) approximation to give a solution to the gap problem. We can simply run the approximation algorithm on the graph and look at the value of the answer. If the value is \( \geq sm \), then we have that \( OPT \geq cm \) else \( OPT \leq sm \). So proving hardness of approximation is proving hardness of gap problems.

The basic idea behind proving hardness of approximation results is thus somehow converting a probabilistic prover/verifier into a graph/structure in our particular setting and then reasoning getting the required gap from the gap between the completeness and the soundness of the prover/verifier.

Before moving ahead we want to make a very important remark, which can be seen as just a continuation of theorem 1.3

**Remark 9.3.** PCP Theorem \( \Leftrightarrow \) there exists a universal constant \( s < 1 \) such that \( \text{GAP} - \text{E3SAT}_{1,s} \) is NP-Hard

## 10 Max-Clique and the FGLSS construction

We now move towards showing our first hardness of approximation result. The problem is referred to as \( \text{GAP} - \text{CLIQUE}_\rho \) and is defined below

**Definition 10.1.** (GAP Clique) \( \text{GAP} - \text{Clique}_\rho \): Given \( \rho \leq 1 \), a graph \( G \) and an integer \( k \), we wish to distinguish between the following two cases

• YES if \( \omega(G) \geq k \)
• NO if \( \omega(G) \leq \rho k \)

where \( \omega(G) \) is the size of the largest clique in the graph.

Equivalently we want to see that can we approximate \( \text{MAXCLIQUE} \) to a factor of \( \rho \). The result is this regard which we are going to prove is the following

**Theorem 10.2.** (GAP-Clique theorem)

• \( \exists \rho < 1 \), such that \( \text{GapCLIQUE}_\rho \) is NP-Hard
• \( \forall \rho > 0 \), \( \text{GapCLIQUE}_\rho \) is NP-Hard

We note that this is not the optimum known and in fact the optimum that is known is really dire and disappointing. First, note that given this problem we can always output just one vertex and feel happy that we have achieved a 1/n approximation. Such an approximation is clearly trivial and our happiness seems unfounded, even absurd. The surprise though is that this is in fact the best we can provably do. That is

**Theorem 10.3.** (GAP-Clique theorem)
\[ \forall \epsilon > 0, \text{GapCLIQUE}_{n^{\epsilon-1}} \text{ is NP-Hard} \]

We now prove theorem 10.2.

Following the general idea of proving hardness of approximation results where in we hope to reduce from the PCP theorem taking advantage of the gap between completeness and soundness, it can be verified such a GAP-preserving reduction can be made from the well-known reduction from SAT to MAXCLIQUE. However what we describe here is a more general construction called the FGLSS graph, named after Feige, Goldwasser, Lovasz, Safra and Szegedy who introduced it in their seminal FOCS paper in 1991 [3]. It is interesting to note that this was in fact the first relation discovered between PCPs and hardness of approximation of specific problems.

### 10.1 FGLSS Construction

Let there be any general PCP verifier \( V \) with \( r \) bits of randomness and constrained to make \( q \) queries. Let us now construct a graph \( G \) which is constructed in the fashion of layers of rows and each row containing some vertices. \( G \) is constructed as

1. There are \( 2^r \) rows, one row corresponding to one instantiation of the \( r \) random coins
2. Each row has \( \leq 2^q \) vertices. Each vertex corresponds to one particular accepting instantiation of the \( q \) bits that the verifier queries for that particular row (i.e. instantiation of the \( r \) bits).
3. Add edges between vertex \( i \) and vertex \( j \) at rows \( r_1 \) and \( r_2 \) if they are consistent with each other (This is done for all \( i, j, r_1, r_2 \)). By consistency \( i \) mean that in the intersection of the two set of bits queried for \( i \) and \( j \), the instantiations represented by the two bits should agree.

The \( G \) constructed above is referred to as the FGLSS graph. We now show the conversion of completeness and soundness to the GAP. Note that if the completeness of the PCP is \( c \), there exists a proof (instantiation of the queried bits) for which more than \( c \cdot 2^r \) instantiations of the random coins accept. Now take such a proof and pick up the related vertices corresponding to each accepting random coin instantiation. Since these instantiations correspond to one common proof, and hence all the picked vertices will be consistent, we see that there is a clique of size \( c \cdot 2^r \).

Now lets see the other case where there exists no proof for which more than \( s \) fraction accept. In this case we see that a clique can only span a maximum of \( s \cdot 2^r \) rows (as otherwise corresponding to the clique we would have a proof that has higher probability of acceptance), and hence the largest clique can be of size \( \leq s \cdot 2^r \).

In summary for a language \( L \) verified by \( V = \text{PCP}_{c,s}[r, q] \), we have that

\[
\begin{align*}
l \in L & \implies \omega(G) \geq c \cdot 2^r \\
l \notin L & \implies \omega(G) \leq s \cdot 2^r
\end{align*}
\]

Note that the size of the graph is \( O(2^r + 2^q) \). Therefore if \( r \) and \( q \) are bounded by \( O(n) \) the graph can be constructed in polynomial time and the reduction is hence done in polynomial time.

Now note that,

\[
\begin{align*}
\text{If } 3\text{SAT} \in \text{PCP1, s}[r, q] \\
\text{then } \phi \in 3\text{SAT} \iff \omega(G) = 2^r \\
\phi \notin 3\text{SAT} \iff \omega(G) \leq s \cdot 2^r
\end{align*}
\]
The above equations make it clear that an \(s - \text{approximation}\) for GAP-CLIQUE leads to a solution for \(3\text{SAT}\), thus proving the NP-HARDness of GAP - CLIQUE. So all that is left to prove is that \(3\text{SAT} \in PCP1, s[r, q]\) for reasonable \(r\) and \(q\). But that is the PCP Theorem that \(NP \in PCP1, s[O(\log(n)), O(1)]\), and hence the first part of theorem 10.2 is proved.

To prove the second part note that using a simple serial repetition \(k\) times on the PCP verifier gives us the following theorem

**Theorem 10.4.** For all \(k\), \(3\text{SAT} \in PCP_{1,1/2^k}[O(k, \log(n)), O(k)]\)

and hence the soundness parameter can be reduced arbitrarily low, using \(O(\log(1/\epsilon))\) many repetitions. This implies that for any \(\forall \rho > 0\), \(\text{GapCLIQUE}_\rho\) is NP-Hard, thus proving theorem 10.2

11 Max-Label-Cover and Raz’s Repetition

We define next the label-cover problem which is referred to in literature as the mother of all ‘sharp’ hardness of approximation results. We note here that Label-Cover and its variants(one of them being leading to the famous Unique Games Conjecture [4]) have been instrumental in proving hardness of a variety of optimization problems including SAT, CLIQUE, MAX-CUT, Vertex Cover, Colouring, MAEX3LIN2 etc. We shall in this section define the label cover problem, followed by giving an equivalent formulation as 2 prover round 1 games, state a very influential and powerful repetition theorem by Raz and then prove the hardness with its help.

11.1 Label Cover

**Definition 11.1.** (Label Cover Problem over \(\sigma\)) The Label-cover with alphabet \(\sigma\) is a constraint graph satisfaction problem with the graph and the constraints bearing the following restrictions:

- The constraint graph is bipartite, \(G = ((V_1, V_2), E)\)
- The graph is left regular, i.e. each vertex on the left has the same cardinality (just a technicality)
- The constraint on an edge \((u, v) = C_{(u,v)}\), is a function from \(\sigma \rightarrow \sigma\) (i.e. there is exactly one accepting label for \(v\) given a label for \(v\)). This is called the projection property.

The major theorem that we are wishing to prove is the following

**Theorem 11.2.** For any \(\epsilon > 0\), there exists a \(\sigma\), such that \(\text{Gap - LabelCover}(\sigma)_{1, \epsilon}\) is NP-Hard

We start off the proof by first giving a small proof proving the hardness for one particular \(\epsilon\). Remember that \(MAXSAT\) is hard to approximate for some \(\epsilon > 0\) by theorem ??

Take a MAX-SAT instance and construct the following graph. The graph is bipartite \(G = (V_1 \cup V_2, E)\). \(V_1\) represents the set of clauses and \(V_2\) represents the set of literals. There is an edge from a clause vertex to a literal vertex if the literal is contained in the clause. The alphabet size is 7. Now think of each of the save labels as the seven different assignments of the literals with which the clause can be satisfied. The labels on the literal vertices mean the true or false assignment on the literal for the first two literals and the other 5 can be discarded. Now we define the constraint which as is hopefully obvious by now is basically the set of consistent labels with respect to the assignment of the clause and the assignment of the literal.

We know that \(MAX - SAT_{1,1-\epsilon}\) is hard to approximate for some \(\epsilon\). Take such an instance and construct the above graph. If we have a fully satisfied instance it is clear that we can label the clauses according to the
satisfying assignment and all edges are satisfied. If we have a maximum \(1 - \epsilon\) satisfying instance, we can only satisfy upto \(1 - \epsilon\) clauses and at least one edge in each of the \(\epsilon\) unsatisfied clauses must be wrong leading to a gap of \(\epsilon/3\).

Therefore we now have a starting point, a constant gap. The idea is to somehow magnify this arbitrarily. Repetitions seem natural and this is what we would do but before that we take a little detour and describe the setting in which the parallel repetition theorem is stated.

### 11.2 2-Prover 1-Round Games

**Definition 11.3.** 2-Prover 1-Round Games A 2P1R game \(G\) is played by two players (or provers) \(P_1\) and \(P_2\) and has the following parameters:

- a set of questions \(X\) for \(P_1\)
- a set of questions \(Y\) for \(P_2\)
- an answer set \(A\)
- a probability distribution \(\lambda\) on \(X \times Y\)
- an acceptance predicate \(V\) on \(X \times Y \times A \times A\)
- a strategy \(f_1 : X \to A\) and \(f_2 : Y \to A\)

The game is played as follows:

- a verifier picks \((x, y) \in X \times Y\) according to \(\lambda\), and asks \(x\) to \(P_1\) and \(y\) to \(P_2\)
- \(P_1\) answers \(a = f_1(x)\) and \(P_2\) answers \(b = f_2(y)\)
- the verifier tests the predicate \(V(x, y, a, b)\); the players \(P_1\) and \(P_2\) win if the predicate is satisfied, and lose otherwise.

We remark here that 2 Prover 1 Round games are essentially equivalent to constraint satisfaction on weighted bipartite graphs. To see this think of \(X\) as \(V_1\), \(Y\) as \(V_2\), \(A\) as the labels, the predicate \(V\) as the constraints, the \(\lambda\) corresponds to the distribution on the edge set which will now be a complete graph between \(V_1\) and \(V_2\).

**Definition 11.4.** Given a 2P1R game \(G\), we say the value of \(G\), denoted by \(\omega(G)\), is the probability (over \(\lambda\)) that \(P_1\) and \(P_2\) win, when they use optimal strategies.

Value corresponds to the maximum fraction of simultaneously satisfiable constraints in a bi-partite constraint graph.

### 11.3 Parallel Repetition

We now describe a form of repetition in the hope that this allows us to decrease the value of a game exponentially. Parallel repetition is the idea of asking many questions but all at once. The main difference between parallel repetition and serial repetition is that in serial repetition the verifier multiple questions but one by one and hence the prover does not have an idea of what questions are coming later. In parallel repetition on the other hand all questions are asked in one go, and hence the prover can see them and then answer them all at once (not necessarily independently).

We define the parallel repetition setup \((G^k)\) for the 2 prover 1 round game \(G\) as follows

---

15
Definition 11.5.  
- $P_1$ has question set $X^k$, and $P_2$ has question set $Y^k$
- the answer set is $A^k$
- the verifier picks $(x_1, y_1), \ldots, (x_k, y_k)$ independently according to $\lambda$
- the verifier sends all $x_1, \ldots, x_k$ simultaneously to $P_1$, and $y_1, \ldots, y_k$ simultaneously to $P_1$.
- the verifier receives answers $a_1, \ldots, a_k$ from $P_1$ and $b_1, \ldots, b_k$ from $P_2$
- $G^k$’s acceptance predicate is $V(x_1, y_1, a_1, b_1) \land \ldots \land V(x_k, y_k, a_k, b_k)$.

It is natural to believe that $\omega(G^k) = \omega(G)^k$. However as a very clever and simple example due to Feige shows, the previous equation does not in general hold. The example is called the Non-interactive Agreement (NA) game.

11.4 The Noninteractive Agreement Game

Definition 11.6. The Noninteractive Agreement 2PRI game
- The verifier flips two independent coins $b_1$ and $b_2$ and sends $b_1$ to $P_1$ and $b_2$ to $P_2$
- Each prover answers an element from the set $\{P_0, P_1\} \times \{0, 1\}$, representing a 'guess' for the coin toss of one of the players. The provers win if there answers are the same and are consistent with the real value.

Observation 1. If the game is being played in one iteration we can see that at least one player has to make a guess about a coin flip he has no idea about. Therefore we see that $\omega(G) \leq 1/2$

We now provide a strategy by which we would have that $\omega(G^2) \geq 1/2$.

The provers have the pre-agreement that $P_0$’s first round coin will equal $P_1$’s second round coin and they work under this assumption. Note that they will win the game if this actually happens and hence the probability of winning would be greater than $1/2$. Now to put this assumption into use they decide to answer $P_0$’s coin flip as the first answer and $P_1$’s coin flip as the second answer. $P_0$ can fill the first answer for himself and the same can be done $P_1$ for the second answer. What is tricky is $P_0$’s second answer and $P_1$’s first answer. But here is where they use the assumption and parallel repetition. The assumption allows $P_0$ and $P_1$ to form an answer given their coins and the parallel repetition specifically allows $P_2$ to form an answer because he has to answer with the same value as his second coin(according to the assumption). This leads to a probability of winning to be $\geq 1/2$.

11.5 Raz’s Parallel Repetition

But what does the above game mean? Does parallel repetition have no guarantee of exponential decay of winning probabilities? Let us inspect the game again and see what happens for k trials. We can think that the provers use the strategy described above for each pair of rounds. Using such a strategy we see that

$$\omega(G^k) \geq (1/2)^{k/2}$$

Feige showed that it is in fact sharp.

Theorem 11.7. Feige’ 91 \[?\] $\omega(G^k) = (1/2)^{k/2}$
So we see that the probability does go down exponentially albeit in a different base. Does that happen in general? Raz puts that question to rest with the following very influential theorem.

**Theorem 11.8. Raz’s parallel repetition [6]** Let $s < 1$ and $\|A\|$ be two constants. There exists $s' < 1$ (only depending on $s$ and $\|A\|$) such that for any 2PR1 game $G$ with answer set $A$ and $\omega(G) = s, \omega(G^k) = (s')^k$, for any $k \geq 1$.

We shall not attempt to prove the theorem here. Now take a label cover instance, think of it as a 2PR1 game, repeat the game parallely and now look at the game as a label cover instance. Note that this new label cover instance has vertices from the set $X^k$ and $Y^k$ and the labelling comes out to be from $\sigma^k$. The constraints are also equivalently defined.

Remember that we had commented that the value of the game is the number of constraints that can be satisfied at the maximum. If we now start with a label cover instance that has a gap of $\epsilon$ (i.e. a maximum of $1-\epsilon$ can be satisfied) then we can amplify that gap exponentially using the above strategy and note that we would need a total blowup of $n^{O(\log(1/\epsilon))}$ and hence the reduction is polynomial.

Therefore we can get arbitrary gaps and so we have proved theorem 11.2

### 12 Hastad’s PCP and the hardness of MAX-E3LIN2

We now define the MAX-E3LIN2 problem

**Definition 12.1. MAXE3LIN2** Given a set linear equations of the form $x_i \oplus x_j \oplus x_k$, over some variable set $X = x_1 \ldots x_n$, we want to output a boolean assignment that satisfies the maximum number of clauses.

We will now prove that the above problem is hard to approximate for any $\epsilon > 0$. To prove this we will prove the following theorem

**Theorem 12.2. Hastad’s PCP** For every $\epsilon, \delta > 0$, $NP \subseteq PCP_{1-\epsilon, 1/2+\delta}[O(\log n), 3]$, where the proofs are binary and every check made by the verifier is of the form XOR of the three bits queried.

We can clearly see that if we evaluate all the equations for $2^{O(\log n)}$ instantiations of the random bits taking the bits of the proof as variables, we get a reduction from SAT using Hastad’s PCP.

So all we need to prove is theorem . Before that some remarks are in order concerning the optimality of this PCP.

**Remark 12.3.** It can be proven that the ratio achieved is indeed optimal, i.e. for any $PCP_{c,s}[O(\log n), 3]$ to contain $NP$, the ratio $s/c \geq 1/2$, unless $P=NP$.

**Remark 12.4.** Also about the completeness, it can be seen that the completeness cannot be increased to 1, keeping the XORing restriction on the verifier, unless $P=NP$.

The idea of the proof of theorem 12 is reduction from Label-Cover. Actually we shall sketch the proof of existence of a machine like the one needed in Hastad’s PCP which can decide GAP-Label Cover. This shall then prove theorem 12 via the hardness of GAP-Label-Cover.

One verification scheme for Label Cover could be to pick an edge randomly and query for the two labels on the ends and check the constraint on this edge. The gap translates as it is. The problem though is the size of the alphabet which has no real bound on the size and also the check which has to be a XOR. The basic idea to achieve this through an appropriate encoding of the proof. We now describe a very celebrated encoding scheme called Long Codes introduced by Goldreich, Bellare and Sudan [95] [8] and provide mechanisms for checking it appropriately in this setting.
Note that the proofs of the theorems that we will now be stating requires fundamentals of fourier analysis on the boolean hypercube and hence we will not be going into them in detail. The interested reader is encouraged to read these set of lecture notes by O’Donell and Guruswami for a good description.

**Definition 12.5.** Long Codes Let the alphabet size be m. The Long Code for a letter i is given by the function \( f_i : \{-1,1\}^m \rightarrow \{-1,1\} \) defined as \( f_i(x) = x_i \). This is also known as the dictator function.

The basic idea behind using this encoding scheme is that any such function holds a lot of information about the letter i that it represents. One other property that is useful is its stability under noise. That is if we choose \( x \in \{-1,1\}^m \) randomly and let y be the result of flipping every bit of x with some probability \( \epsilon \), then \( \Pr[f(x) \neq f(y)] = \epsilon \). Note that the dictator function is linear.

The test combines ideas from Linearity Testing with the stability under noise feature to filter out other linear functions passing the test. The test is given in the following

**Definition 12.6.** Formally, the verifier picks an edge \((u,v)\) randomly. The prover is now supposed to provide the long codes for all the labels on the vertices. Let A and B be the long codes of the labels advertised by the prover. The verifier also selects a random \( x \) and \( y \in \{-1,1\}^m \) and a noise vector \( \mu \in \{-1,1\}^m \) with \( \epsilon \) bias and then checks the following property

\[
A(x)B(y)B((x \times \pi)y\mu) = 1
\]

where \( xy \) is vector representing component wise multiplication between x and y and \( x \times \pi \) is the rearrangement of the values of x according to the permutation \( \pi \).

To see the completeness, we can observe that if the prover actually writes the long codes in the proof all values queried would be what they should be and as such the constraint would be correctly checked. This implies that if the instance is fully satisfiable we can actually write the proofs correctly and get a valid solution and hence the completeness is 1.

We now state without proof that the soundness of the above procedure is infact 1/2. The proof can be found here. [7]. What we do want to highlight is the fact that there are two kinds of checks that are required, one is that the code given by the prover is in fact a valid Long Code and also somehow include the check for the constraint in it. In the above we used a strong notion of adding some noise to the check, which is in fact one of the central methods to check Long Codes as well as other codes too.

### 13 Conclusion

We have in the paper defined the PCP theorem, described a very illuminating proof and then showed its impact on hardness of approximation. It is interesting to note that building on these ideas there have been very many hardness of approximation results, some of them very general and strong. It has been one of the highly researched areas of Computational Complexity over the last two decades and still has a lot of potential for throwing new ideas as the recent research over the unique games conjecture has suggested.

### References


[7] Ryan O’Donnel, V. Guruswami Lecture notes of The PCP Theorem and Hardness of Approximation The PCP Theorem and Hardness of Approximation, Autumn 2005

