Probabilistically Checkable Proofs: A Primer

Madhu Sudan*

July 11, 2006

Abstract

Probabilistically checkable proofs are proofs that can checked probabilistically by reading very few bits of the proof. Roughly ten years back it was shown that proofs could be made probabilistically checkable with a modest increase in their size. While the initial proofs were a little too complex, a recent proof due to Irit Dinur gives a dramatically simple (and radically new) construction of probabilistically checkable proofs. This article explains the notion, presents the formal definition and then introduces the reader to Dinur's work and explains some of the context (but does not reproduce Dinur's proof).

1 Introduction

As advances in mathematics continue at the current rate, editors of mathematical journals increasingly face the challenge of reviewing increasingly long, and often wrong, "proofs" of classical conjectures. Often, even when it is a good guess that a given submission is erroneous, it takes excessive amounts of effort on the editor/reviewer's part to find a specific error one can point to. Most reviewers assume this is an inevitable consequence of the notion of verifying submissions; and expect the complexity of the verification procedure to grow with the length of the submission. The purpose of this article is to point out that this is actually not the case: There does exist a format in which we can ask for proofs of theorems to be written. This format allows for perfectly valid proofs of correct theorems, while any purported proof of an incorrect assertion will be "evidently wrong" (in a manner to be clarified below). We refer to this format of writing proofs as Probabilistically Checkable Proofs (PCPs).

In order to formalize the notion of a probabilistically checkable proof, we start with a bare-bones (computationally simplified) view of logic. A system of logic is described by a collection of axioms which include some "atomic axioms" and some derivation rules. An *assertion* is a sentence, which is simply a sequence of letters over the underlying alphabet. A *proof* of a given assertion is a sequence of sentences ending with the assertion, where each sentence is either one of the axioms or is obtained by applying the derivation rules to the previous sentences in the proof. An assertion which has a proof is a *theorem*. We will use the phrase *argument* to refer to sequence of sentences (which may be offered as "proofs" of "assertions" but whose correctness has not been verified).

*CS & AI Laboratory (CSAIL), Massachusetts Institute of Technology, 32-G640, 32 Vassar Street, Cambridge, MA 02139, USA. http://theory.csail.mit.edu/~madhu. This article supported in part by NSF Award CCR-0312575. Views expressed in this article are those of the author, and not endorsed by NSF.

While systems of logic come in many flavors and allow varying degrees of power in their inference rules and the nature of intermediate sentences that they would allow, the "computational perspective" unifies all of these by using the following abstraction: It suggests that a system of logic is given by a computationally *efficient* algorithm called the *verifier*. The inputs to a verifier is a pair of sequences over some finite alphabet, an assertion T and evidence Π and accepts this pair if and only if Π forms a proof of T in its system of logic. Such verifiers certainly capture all known systems of logic. Indeed without the computational efficiency restriction, it would be impossible to capture the spirit that theorems are often *hard* to prove, but once their proofs are given, they are *easy* to verify. For our purposes, we associate the word "efficient" with the feature that the algorithm runs in time polynomial in the length of its inputs. (As an aside, we note that this distinction between the proving theorems and verifying proofs is currently a conjecture, and is exactly the question examined under the label "Is P=NP?".)

The notion that a verifier can perform any polynomial time computation enriches the class of theorems and proofs considerably and starts to offer highly non-trivial methods of proving theorems. (One immediate consequence is that we can assume theorems/proofs/assertions/arguments are *binary* sequences and we will do so henceforth.) For instance, suppose we have an assertion A (say the Riemann Hypothesis), and say we believe that it has proof which would fit within a 10,000 page article. The computational perspective says that given A and this bound (10,000 pages), one can efficiently compute three positive integers N, L, U with $L \leq U \leq N$ such that A is true if and only if N has a divisor between L and U. The integers N, L, and U will be quite long (maybe writing them would take a million pages), yet they can be produced extremely efficiently (in less than the amount of time it would take a printer to print out all these integers, which is certainly at most a day or two). (This example is based on a result due to Joe Kilian, personal communication.) The theory of NP-completeness could be viewed as an enormous accumulation of many other equivalent formats for writing theorems and proofs. Depending on one's perspective, this may or may not be a better format for writing theorems and proofs. What is important for us is that despite the fact that it differ radically from our mental picture of theorems/proofs - this is as valid a method as any. Every theorem has a valid proof, and this proof in only polynomially larger than the proof in any other system of logic, a notion referred to as "completeness". Conversely, no false assertion has a proof, a notion referred to as "soundness".

The ability to perform such non-trivial manipulations to formats in which theorems and proofs are presented raises the possibility that we may specify formats that allow for other features (that one does not expect from classical proofs). The notion of PCPs emerges from this study. Here we consider verifiers that vary in two senses: (1) The verifiers are probabilistic — they have access to a sequence of unbiased independent coins (i.e., random variables taking on values from the set $\{0, 1\}$); and (2) The verifiers have "oracle" access to the proof. I.e., to read any specific bit of the proof the verifier is allowed direct access to this bit and charged one "query" for this access. (This is in contrast to the classical notion of the Turing machine where all information is stored on tapes and accessing the ith bit takes i units of time and implies access to all the first i bits of the proof.) However, we will restrict the number of random bits that the verifier has access to. We will also restrict the number of queries the verifier is allowed to make. The latter is definitely a restriction on the power of the verifier (classical verifiers accessed every bit of the proof). The former does not enhance the power of the verifier *unless* the verifier is allowed to err. So we will allow the verifier to err and consider the question: It must be stressed at this point that we require the error probability is bounded away from 1 for *every* false assertion and *every* supporting argument. (It would not make any sense, given the motivation above to assume some random distribution over theorems

and proofs, and this is not being done.) What is the tradeoff between the query complexity and the error incurred by the verifier?

Theoretical computer scientists started to examine this tradeoff starting 1990 and have made some remarkable progress to date. We review this history below. (We remark that this is just a history of results; the notion of a probabilistically checkable proof itself evolved slowly over a long sequence of works [17, 6, 9, 15, 4, 14, 3], but we will not describe the evolution of this notion here.) Results constructing PCP verifiers typically restrict the number of random bits to be logarithmic in the size of the probabilistically checkable proof. Note that this is an absolute minimum limit, or else a verifier making few queries does not have a positive probability of accessing most of the bits of the proof. They then asked the question: How small the can the PCP be (relative to the classical proof) and how many bits needed to be queried? The first sequence of results [5, 4, 14] quickly established that the number of queries could be exponentially smaller than the length of the proof (e.g., in a proof of length n, the number of queries may be as small as say $\log^2 n$), while getting nearly polynomial sized proofs (in fact, [4] obtained nearly linear sized PCPs.) The second short sequence [3, 2] established what is now referred to as "The PCP Theorem" which showed that the number of bits queried could be reduced to an absolute constant(!) independent of the length of the theorem or the proof (given just the length of the proof), with PCPs of length just a polynomial in the classical proof. This immediately raised the question: What is this universal constant — the number of queries that suffices to verify proofs probabilistically. It turns out there is yet another tradeoff hidden here. It is always possible to reduce the number of queries to three bits, if the verifier is allowed to err with probability very close to (but bounded away from) one. So to examine this question, one needs to fix the error probability. So, say we insist that arguments for incorrect assertions are accepted with probability (close to) half, while proofs of valid theorems are accepted with probability one. In such a case, the number of queries made by the verifier of [2] has been estimated at around 10^6 bits - not a dramatically small constant, though a constant all right! The third phase in the construction of PCPs [8, 7] attempted to reduce this constant and culminated in yet another surprise. Hastad [18] shows that the query complexity could be essentially reduced to just *three* bits to get the above error probabilities. Subsequent work in this area has focussed on the question of the size of the PCP relative to the size of the classical proofs and shown that these could be reduced to extremely blow-ups. (Classical proofs of length n are converted to PCPs of length $n \cdot (\log n)^{O(1)}$ in the work of Dinur [12].)

A somewhat orthogonal goal of research in PCPs has been to find simple reasons why proofs ought to be probabilistically checkable. Unfortunately, much of the above results did not help in this regard. The results from the first sequence achieved the effect by a relatively straightforward but striking algebraic transformation (by encoding information into values of algebraic functions over finite fields). Later results built on this style of reasoning but got even more complex (see e.g., [25, Page 12] for a look at the ingredients needed to get the PCP theorem of [18]). Recently, Dinur and Reingold [13] proposed a novel, if somewhat ambitious, iterative approach to constructing PCPs, which was radically different than prior work. While the idea was appealing, the specific implementation was still hard, and did not lead to a satisfactory alternative construction of PCPs. Subsequently, Dinur [12] finally made remarkable progress on this question deriving the right ingredients to give a dramatically simple proof of the PCP theorem.

This work of Dinur is the focus of the rest of this article. Our intent, however, is not to give Dinur's proof of the PCP theorem. This is already done quite satisfactorily in her work [12]. Instead we will try to outline her approach and provide context to the steps taken in Dinur which may provide further insight into her work (and highlight the novelty of the approach as well as the new technical

ingredients developed in her work). The hope is that a reader, after reading this article, would be motivated to read the original work, and upon doing so, appreciate the developments in her paper.

In what follows, we will start by formally describing PCPs and the PCP theorem. Readers uncomfortable with boring formalisms could skip this section. Next we describe a duality between PCP verifiers and "approximations to combinatorial optimization problems". We will use this duality to switch our language from the "logical" theme of theorems and proofs, to a more "combinatorial" theme. (A reader who chooses to skip this section would be lost thereafter.) In Section 4 we then describe the high-level approach in Dinur's paper and contrast it with the earlier approaches. Dinur's approach repeatedly applies two transformations to a "current verifier", starting from a classical (non-probabilistic) verifier of proofs. The end result is a probabilisitic verifier of proofs. In Sections 5 and 6 we describe the two transformations in greater detail providing background on these (in particular, we describe some simpler transformations one may consider, and why they don't work).

2 Definitions and formal statement of results

We start by recalling the notion of a classical verifier and introducing some notation.

First some general notation for the paper. Below $\mathbb{R}$ will denote the reals, $\mathbb{Z}$ the set of all integers, and $\mathbb{Z}^+$ the set of positive integers. For $x \in \mathbb{R}$, we let $\lfloor x \rfloor$ denote the largest integer less than or equal to x. For $x \in \mathbb{R}$, let $\log x$ denote the quantity $\lceil \log_2 x \rceil$ where $\log_2$ denotes the logarithm of x to base 2.

By $\{0,1\}^*$ we denote the set of all finite length binary sequences. (We refer to such sequences as strings.) For a string $x \in \{0,1\}^*$, let $|x|$ denote its length. For random variable X taking on values in domain D and event $E : D \in \{\text{true}, \text{false}\}$, we let $\Pr_X[E(X)]$ denote the probability of the event E over the random choice of X. We often use the shorthand "$f(n)$" to denote the function $n \mapsto f(n)$. (In particular, it will be common to use "n" to denote the argument of the function, without explicitly specifying so.) Examples include the functions n^2, $\log n$ etc.

Later in the writeup we will need to resort to some "graph theory". By a graph we refer to symmetric pairwise relationships on some finite set. Formally a graph G is given by a pair (V, E) with V being a finite set and $E \subset V \times V$ is a symmetric relationship. If $(u, v) \in E$, then we refer to v as being adjacent to u, or being a neighbor of u. The number of vertices adjacent to u is called the degree of u. We say a graph has degree D if every vertex has degree D. A walk in a graph is a finite sequence of vertices $v_0, \ldots, v_\ell$ such that v_{i-1} and v_i are adjacent for every $i \in \{1, \ldots, \ell\}$. The distance between u and v is the length ℓ of the shortest walk $v_0, \ldots, v_\ell$ satisfying $v_0 = u$ and $v_\ell = v$.

We now move to notions related to proof verification. A *verifier* is a polynomial time algorithm computing a function $V : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, with the association that $V(T, \Pi) = 1$ implies that the assertion T is a theorem with Π being a proof. (Recall that a function is said to be polynomial time computable if there exists an algorithm running in time bounded by a fixed polynomial in the total length of its inputs to compute the function.) Given a polynomial $p : \mathbb{Z}^+ \to \mathbb{Z}^+$ and verifier V, let $L_{V,p}$ denote the set of theorems with "short" proofs of length at most $p(n)$. I.e., $L_{V,p} = \{T \in \{0,1\}^* | \exists \Pi \in \{0,1\}^{p(|T|)} \text{ s.t. } \mathrm{V}(\mathrm{T}, \Pi) = 1\}$. The class NP is the set of all such sets $\{L_{V,p} | V$ is a verifier and p is a polynomial $\}$.

As mentioned earlier, we are going to enhance classical algorithms by endowing them with access to random strings and oracles. We will denote random strings just like other strings. A oracle will just be a function $O : \mathcal{Q} \to \mathcal{A}$ where $\mathcal{Q}$ is a countable set and $\mathcal{A}$ is finite. The most common version is with $\mathcal{Q} = \mathbb{Z}^+$ and $\mathcal{A} = \{0,1\}$. Algorithms are allowed to compute various queries $q_1, \ldots, q_t$ and obtain answers $O[q_1], \ldots, O[q_t]$ to the queries. The number of queries made (t) is termed the query complexity of the algorithm. Thus the computation of a probabilistic oracle algorithm A on input x, random string $R \in \{0,1\}^*$ and access to oracle O will be denoted $A^O(x;R)$. Notice that we will always be interested in the distribution of this random variable $A^O(x;R)$ when R is chosen uniformly from set $\{0,1\}^\ell$ (while x and O will be fixed). With this notation in hand we are ready to define PCP verifiers and the complexity class PCP.

Definition 1 *For functions $r, q : \mathbb{Z}^+ \to \mathbb{Z}^+$ an (r,q,a)-restricted PCP verifier is a probabilistic oracle algorithm V that on input $x \in \{0,1\}^n$, expects a random string $R \in \{0,1\}^{r(n)}$ and queries an oracle $\Pi : \mathbb{Z}^+ \to \{0,1\}^{a(n)}$ at most $q(n)$ times and computes a "Boolean verdict" $V^\Pi(x;R) \in \{0,1\}$.*

Definition 2 *For functions $c, s : \mathbb{Z}^+ \to [0,1]$ with $0 \leq s(n) < c(n) \leq 1$ for every $n \in \mathbb{Z}^+$, we say that an (r,q,a)-restricted PCP verifier V accepts a set $L \subseteq \{0,1\}^*$ with completeness c and soundness s if for every $x \in \{0,1\}^n$ the following hold:*

Completeness: *If $x \in L$ then there exists a $\Pi : \mathbb{Z}^+ \to \{0,1\}^{a(n)}$ such that $\Pr_R[V^\Pi(x:R) = 1] \geq c(n)$.*

Soundness: *If $x \notin L$ then for every $\Pi : \mathbb{Z}^+ \to \{0,1\}^{a(n)}$ it is the case that $\Pr_R[V^\Pi(x:R) = 1] \leq s(n)$. By $PCP_{c,s}[r,q,a]$ we denote the class of all sets L such that there exists an (r,q,a) restricted PCP verifier accepting L with completeness c and soundness s.*

Throughout this article we will assume that the queries of the PCP verifiers are made "non-adaptively". I.e., the exact location of questions does not depend on the responses to other questions. The responses only affect the accept/reject predicate of the verifier.

As described above the class PCP is significantly over parametrized. These different parameters are useful when describing various (steps in) constructions of PCPs, but for now they are likely to burden the reader. So lets discard a few to derive a simpler collection: All early PCP results were phrased in terms of verifiers that achieved perfect completeness $c(n) = 1$; and soundness $s(n) \leq \frac{1}{2}$. They also fixed $a(n) = 1$ — i.e., oracles responded with one bit per query. Letting $\mathrm{PCP}[r,q]$ denote the class of languages with such restrictions, the early results [5, 4, 14] could be described as showing that there exist polynomials $p_1, p_2 : \mathbb{Z}^+ \to \mathbb{Z}^+$ such that $\mathrm{NP} \subseteq \mathrm{PCP}[p_1(\log n), p_2(\log n)]$. The PCP Theorem, whose new proof we hope to outline later, may now be stated formally as.

Theorem 3 ([3, 2]) *There exist a constant q such that $NP = \cup_{c \in \mathbb{Z}^+} PCP[c \log n, q]$.*

Finally, the state of the art result along these lines is that of Håstad [18], which shows that for every $\epsilon > 0$, $\mathrm{NP} = \cup_{c \in \mathbb{Z}^+} \mathrm{PCP}_{1-\epsilon, \frac{1}{2}+\epsilon}[c \log n, 3]$.

One aspect we do not dwell on explicitly is the size of the "new proof". It is easy to convert an (r,q)-PCP verifier into one that runs in time $2^{r(n)} \times 2^{q(n)}$, whose queries are always in the range $\{1, \ldots, 2^{r(n)+q(n)}\}$. In other words one can assume "w.l.o.g." that the proof is a string of size at most $2^{r(n)+q(n)}$. So in particular if the randomness and query complexity are bounded by $O(\log n)$, then the PCP proofs are still polynomial sized, and so we won't worry about the size explicitly.

3 Optimization and approximation

As alluded to earlier, one of the principal motivations for studying proofs from a computational perspective is that they shed light on the tractability of many computational tasks. For instance, the theory of NP-completeness says that a vast collection of combinatorial optimization problems, such as the "Travelling Salesman Problem" (TSP), (given an $n \times n$ matrix of distances between n cities, find the smallest tour that visits all n cities), the "Independent Set Problem" (given a set of n elements and a list of incompatible pairs, find the largest subcollection that consists no pair of incompatible elements) or the "Knapsack Problem" (given the weights and values of n elements and a bound C, find a subset of elements whose weight sums to less than C, while maximizing the sum of their values), the theory of NP-completeness shows that finding optimal solutions is as hard as finding "proofs" for generic theorems. Formally, given any assertion and a bound on the length B of its proof, one can construct an instance of the NP complete problem, say TSP, and an integer B', such that any solution to the TSP of length at most B' implies that the given assertion is true and has a proof of length at most B. Thus an algorithm to find optimal tours is a generic theorem prover which needs to only know the length of the proof to generate the proof. Indeed much of the strength for the belief that P $\neq$ NP may be attributed to the belief that we don't expect theorem-proving to be automated.

In this context it may make sense that a "probabilistic" notion of checking proofs may lead to some further insight on the complexity of solving combinatorial optimization problems. This guess turns out be true, and it turns out that the existence of PCP verifiers implies that for many of these optimization problems finding "nearly optimal" solutions is as hard as finding optimal solutions. This connection was first made by Feige et al. [14], who showed that the PCP theorem (then still a conjecture) would imply that the independent set size could not be approximated to within constant factors. Subsequently, many other optimization problems were shown to be hard to approximate using the PCP theorem (cf. [2, 21]).

Remarkably, Irit Dinur's proof uses a "folklore" reverse connection which shows that "reductions" showing hardness of approximating some optimization problems can a folklore one) that shows that "hardness of approximating yield PCP verifiers. We describe the optimization problem, used in her proof next, and then explain why the inapproximability of this problem yields a PCP verifier next.

Definition 4 (Constraint Satisfaction Problem (Max k-CSP-Σ**))** *For a finite set Σ and integer k, an input to the problem* Max k-CSP-Σ *consists of m constraints $C_1, \ldots, C_m$ on n variables $X_1, \ldots, X_n$, where a constraint C_j consists of a function $f_j : \Sigma^k \to \{0,1\}$ and k indices $i_1(j), \ldots, i_k(j) \in \{1, \ldots, n\}$. An assignment $\langle X_1, \ldots, X_n \rangle \leftarrow \langle a_1, \ldots, a_n \rangle \in \Sigma^n$* satisfies *the constraint C_j if $f_j(\alpha_1, \ldots, \alpha_k) = 1$ where $\alpha_\ell = a_{i_\ell(j)}$. The goal is to compute an assignment, given $C_1, \ldots, C_m$, that maximizes the number of constraints that are satisfied.*

Since Max k-CSP-Σ occupies a central role in this article, let us introduce some notation that will be useful later. We often use ϕ to denote instances of Max k-CSP-Σ and $\vec{a} \in \Sigma^n$ to denote an assignment to the n variables. For a pair $\phi, \vec{a}$ as above, we use the notation $\phi(\vec{a})$ to denote the number of constraints of ϕ satisfied by $\vec{a}$. An instance ϕ of Max k-CSP-Σ is said to be *satisfiable* if there exists an assignment satisfying all constraints. The unsatisfiability of the instance ϕ, denoted UNSAT(ϕ), is the quantity $\min_{\vec{a}}\{1 - \phi(\vec{a})/m\}$ i.e., the minimum fraction of constraints left unsatisfied by any assignment.

Max k-CSP-Σ problems arise naturally in the theory of NP completeness. The classical 3SAT problem is easily captured as an instance of Max3-CSP-$\{0,1\}$ where the goal is to distinguish satisfiable instances from instances that are not satisfiable. Similarly, the classical 3-coloring problem (given a graph on n vertices, determine if it is possible to color the vertices with three colors $\{R,G,B\}$ such that no edge of the graph is monochromatic), can also be expressed as an instance of Max2-CSP-$\{R,G,B\}$. Indeed it is a classical result [16] that for every $k \geq 2$ and every Σ with $|\Sigma| \geq 2$, it is NP-hard to find optimal solutions to Max k-CSP-Σ.

However, the classical result does not say anything about solving the problem near-optimally. In particular, the state of knowledge prior to the PCP theorem allowed for the possibility that some polynomial algorithm could, on input ϕ for which there exists an assignment satisfying t out of m constraints, always produce an assignment satisfying $t(1-o(1))$ constraints! Indeed this may be a good point to introduce the notion of an approximation algorithm.

Definition 5 *For $\alpha \geq 1$ An algorithm A that takes as input an instance ϕ and in polynomial time outputs an assignments $\vec{a} \in \Sigma^n$ such that $\phi(\vec{a}) \geq \phi(\vec{a}')/\alpha$ for every other assignment $\vec{a}' \in \Sigma^n$ is called an α-approximation algorithm for* Max k-CSP-Σ.

The PCP theorem rules out the possibility of α-approximation algorithms for Max k-CSP-Σ, unless NP=P. The following proposition gives a weak version of this result.

Proposition 6 *Let q be the constant from Theorem 3. If there is a $(2-\epsilon)$-approximation algorithm for* Max$q$-CSP-$\{0,1\}$ *for any $\epsilon > 0$, then P=NP.*

Proof of Sketch: Let A be a $2-\epsilon$ approximation algorithm for Max$q$-CSP-$\{0,1\}$. Let L be a language in NP we wish to decide. Let $V = V_L$ be the $(r(n) = O(\log n), q)$-PCP verifier for this language as guaranteed by Theorem 3. Now consider a string $x \in \{0,1\}^n$ for which we wish to know if $x \in L$ or not. Let $\ell \leq 2^{r(n)+q}$ denote the size of the PCP proof that V queries to check membership of $x \in L$. Denote by $X_1, \ldots, X_\ell$ the Boolean variables representing the oracle responses to the verifiers queries. Now for each random string $R \in \{0,1\}^{r(n)}$ create a q-ary constraint C_R as follows: Let $i_1, \ldots, i_q$ be the q queries made by V on input x and random string R. Furthermore, let $f = f(A_1, \ldots, A_q)$ denote the verifier's acceptance predicate on responses A_t to query i_t. Let $C_R = (f, (i_1, \ldots, i_q))$ be the Rth constraint. Let $\phi = \langle C_R \rangle_{R \in \{0,1\}^{r(n)}}$ be the instance of Max$q$-CSP-$\{0,1\}$ thus obtained.

It is easy to verify that $\mathrm{UNSAT}(\phi) = 1 - \max_\Pi \{\Pr_R[V^\Pi(x;R)]\}$. Thus, if $x \in L$ then ϕ is satisfiable and if $x \notin L$ then $\mathrm{UNSAT}(\phi) \geq \frac{1}{2}$. Now consider running A on ϕ. If $x \in L$, then $A(\phi)$ produces an assignment satisfying $m/(2-\epsilon)$ constraints, where $m = 2^{r(n)}$. On the other hand, if $x \notin L$, no assignment satisfies more than $m/2$ constraints. Thus to decide if $x \in L$, all we need to do is to count the number of assignments satisfied by $A(\phi)$ and accept iff this number is more than $m/2$. Since the transformation of x to ϕ takes only polynomial time, and A runs in polynomial time, this gives a polynomial time algorithm to solve a generic NP language L, thus yielding NP=P. ∎

We use the phrase "inapproximable to within a factor of α" to denote that existence of an α-approximation algorithm would imply $P = NP$.

The above proposition and proof only cover the case of Max k-CSP-Σ for some choice of k and Σ. However standard reductions can then be used to show that Max k-CSP-Σ is inapproximable to

within some constant $\alpha > 1$ for every $k \geq 2$ and every Σ with $|\Sigma| \geq 2$. We will elaborate on this later.

The proof above shows that to show that Max k-CSP-Σ is α-inapproximable, it suffices to produce a reduction of the following form for some NP complete language L: The reduction should map, in polynomial time, an instance $x \in \{0,1\}^n$ to an instance ϕ of Max k-CSP-Σ such that ϕ is satisfiable if $x \in L$ and $\text{UNSAT}(\phi) \geq 1 - \frac{1}{\alpha}$. The following proposition shows that any such reduction implies the PCP theorem.

Proposition 7 *Suppose there is a polynomial time reduction from an NP complete language L to* Max k-CSP-Σ *mapping an instance x to ϕ such that ϕ is satisfiable if $x \in L$, and* $\text{UNSAT}(\phi) \geq \epsilon$ *if $x \notin L$. Then $L \in PCP_{1,1-\epsilon}[O(\log n), k, \log |\Sigma|]$.*

Proof of Sketch: The verifier for the assertion "$x \in L$" uses the reduction to produce an instance ϕ of Max k-CSP-Σ. It then expects as proof an oracle Π giving the assignment satisfying ϕ. (So $\Pi[i] = a_i$ where $\vec{a} = \langle a_1, \ldots, a_n \rangle$ is the assignment satisfying ϕ.) To verify the proof, the verifier picks a random constraint C_j of ϕ and verifies it is satisfied by Π. Notice thus that the verifier makes k queries to the proof oracle, getting an element of Σ (which can be encoded by $\log |\Sigma|$ bits) as response. It can also be verified that the verifier accepts with probability one if $x \in L$ and with probability at most $1 - \epsilon$ if $x \notin L$. ∎

Dinur's proof directly produces a reduction showing such a hardness. We state her main theorem below.

Theorem 8 ([12]) *There exists an NP complete language L, $\epsilon > 0$, finite set Σ, and a polynomial time reduction R, mapping instances x to ϕ of* Max2-CSP-Σ *such that ϕ is satisfiable if $x \in L$ and* $\text{UNSAT}(\phi) \geq \epsilon$ *if $x \notin L$.*

Combined with Proposition 7 above, this yields the PCP theorem.

4 Overview of Dinur's approach

Before moving on to describing Dinur's approach to proving the PCP theorem, let us briefly describe the prior approaches. The prior approaches to proving the PCP theorem were typically stated in the "$\text{PCP}_{c,s}[r, q, a]$" notation, but the effective equivalence with Max k-CSP-Σ allows us to interpret them in the CSP notation, and we do so below.

One of the principal issues to focus on is the "Gap" in the unsatisfiability achieved by the reduction. Notice that the reductions we seek achieve a significant gap in the unsatisfiability of the instances achieved when $x \in L$ (which should be 0) and the unsatisfiability when $x \notin L$ (which should be lower bounded by some absolute constant $\epsilon > 0$). We refer to this quantity as the "Gap" of the reduction.

Previous approaches were very careful to maintain large gaps in reductions. Since it was unclear how to create a direct reduction from some NP complete language L to Max k-CSP-Σ for finite k and Σ with a positive gap, the prior approaches considered allowing k and Σ to grow with $n = |x|$. The results of Babai et al. [5, 4] and Feige et al. [14] used algebraic techniques (representing

information as coefficients of multivariate polynomials and encoding them by their evaluations) to get reductions from any NP-complete language L to Max$k(n)$-CSP-$\Sigma(n)$ where $k(n), \log|\Sigma(n)| \approx (\log n)^{O(1)}$. Arora and Safra [3], observed an asymmetry in the behavior of the two parameters $k(n)$ and $\log|\Sigma(n)|$ and in particular observed that one could interpret existing PCP constructions as techniques that reduce Max k-CSP-$\Gamma(n)$ to Max$O(k)csp\Sigma(n)$ where $|\Sigma(n)| \ll |\Gamma(n)|$.

This motivated the search for new PCPs which maintained k to be some absolute constant, while allowing $\Sigma(n)$ to grow. Arora et al. [2] produced two such reductions, one of which reduced Max k-CSP-$\Gamma(n)$ to Max$O(k)$-CSP-$\Sigma(n)$ with $\log|\Sigma(n)| \approx (\log\log|\Gamma(n)|)^3$, and another reduction reducing Max k-CSP-$\Sigma(n)$ to Max$O(k)$-CSP-$\{0,1\}$ (but with the catch that the reduction took time that was at least $\Sigma(n)$, so one couldn't afford to use it on large $\Sigma(n)$). Each one of these reductions reduced the gap by a constant factor, but this was ok since one only needed to apply these reductions a constant number (thrice in [2]) to reduce the alphabet size $\Sigma(n)$ to an absolute constant.

Thus the previous approach could be described as constructing PCPs by "alphabet reduction", subject to "gap preservation". In contrast, Dinur's approach seems to be quite the opposite. In her approach, she starts with a reduction from the NP complete language L to Max k-CSP-Σ which has minimal gap (producing only UNSAT$(\phi) \geq 1/m$ when $x \notin L$), but where k and Σ are finite. She then applies a sequence of iterations that ensure "gap amplification" while "preserving alphabet size". The following lemma, from which the main theorem follows easily describes the properties of these iterations.

Lemma 9 (Main Lemma) *There exists a finite set Σ, a positive constant $\epsilon > 0$ and a linear time reduction*[1] *T transforming instances of* Max 2-CSP-Σ *to instances of the same problem such that*

Completeness *ϕ is satisfiable $\Rightarrow$ $T(\phi)$ is satisfiable.*

Soundness UNSAT$(\phi) \geq \min\{2\text{UNSAT}(\phi), \epsilon\}$.

The reduction above is totally novel in the PCP literature and already finds other applications (other than providing alternate proofs of the PCP theorem) in Dinur's paper (see [12, Section 7]). Indeed a few iterations (logarithmically many) of the transformation above amplifies the gap of any reduction from tiny amounts to an absolute constant, and thus yields the PCP theorem. The following proof argues this formally.

Proof of of Theorem 8: Given an NP complete language L and a string $x \in \{0,1\}^n$ for which we wish to decide membership, we first transform it to an instance ϕ_0 of Max 2-CSP-Σ such ϕ_0 is satisfiable if and only if $x \in L$. (Notice such reductions, with effectively trivial gap, are classical.) Let m denote the number of constraints of ϕ_0. Now iterate the transformation T from Lemma 9 $\ell = \log m$ times, and let $\phi_i = T(\phi_{i-1})$. We claim that the reduction that maps x to ϕ_ℓ has the properties claimed in the theorem.

First note that if $x \in L$, then ϕ_i is satisfiable for every $i \in \{0, \ldots, \ell\}$ satisfying the "completeness" condition.

[1] I.e., there exist absolute constants c, d such that $T(\phi)$ takes time at most $c|\phi| + d$ to compute. In particular, this implies that $|T(\phi)| \leq c|\phi| + d$.

Next note that if $x \notin L$, then $\text{UNSAT}(\phi_0) \geq 1/m$ (since ϕ_0 is not satisfiable). By induction (using the Soundness condition in Lemma 9) we can now see that $\text{UNSAT}(\phi_i) \geq \min\{\frac{2^i}{m}, \epsilon\}$. Thus, since $2^\ell \geq m$, we have $\text{UNSAT}(\phi_\ell) \geq \epsilon$.

Finally, we need to argue that the entire reduction takes polynomial time. To do this it suffices to argue that the size of the instance ϕ_ℓ is only polynomially larger than x. (The total running time is then bounded by the time take to produce ϕ_0 plus at most $\log m$ times some linear function in $|\phi_\ell|$.) To argue this we use (in fact, need!) the fact that T is a linear time reduction and so $|T(\phi)| \leq c|\phi| + d$. For simplicitly, assume $d = 0$. Then by induction, we see that $|\phi_\ell| \leq c^\ell \cdot |\phi_0| \leq O(m^{\log_2 c}) \cdot |\phi_0| \leq (|\phi_0|)^{O(1)} \leq (|x|)^{O(1)}$ as required. ∎

Thus our focus now shifts to Lemma 9 and we start to peek into its proof. Dinur's proves this Lemma by combining two counteracting reductions. The first reduction amplifies the gap by increasing the alphabet size. Since this is the main novelty in Dinur's reduction, we will defer its proof to the end. The second reduction is now in the classical style, which reduces the gap (somewhat), while reducing the alphabet size. While it is clear that both reductions are opposing in direction, the level of detail used above leaves it unclear as to what would happen if the two reductions were applied in sequence. Would this increase the gap or reduce it? Would it increase the alphabet size or reduce it (or preserve it)?

Part of the insight behind Dinur's approach is the observation that both these reductions are especially strong. The first allows gap amplification by any amount, subject to a sufficiently large explosion in the alphabet size. The second reduction can reduce any alphabet to a fixed small alphabet, while paying a fixed price in terms of the gap. These terms are articulated in the assertions below.

Lemma 10 *For every constant $c < \infty$ and finite set Σ there exist constant $\epsilon_1 > 0$, finite set Γ and a linear time reduction T_1 from* Max 2-CSP-Σ *to* Max 2-CSP-Γ *such that:*

Completeness *ϕ is satisfiable $\Rightarrow$ $T_1(\phi)$ is satisfiable.*

Soundness $\text{UNSAT}(T_1(\phi)) \geq \min\{c \cdot \text{UNSAT}(\phi), \epsilon_1\}$.

(In other words, one can pick any amount to amplify by, and the reduction finds an appropriate alphabet Γ to reduce to.)

Lemma 11 *There exists a constant $\epsilon_2 > 0$, a finite set Σ such that for every finite Γ, there exists a linear time reduction T_2 mapping* max 2*cspg to* max 2*csps such that:*

Completeness *ϕ is satisfiable $\Rightarrow$ $T_2(\phi)$ is satisfiable.*

Soundness $\text{UNSAT}(T_2(\phi)) \geq \epsilon_2 \cdot \text{UNSAT}(\phi)$.

Notice that ϵ_2 above — the loss in the gap — is independent of alphabet size. We will elaborate more on this in the next section.

We defer the proofs of the two lemmas to the ensuing sections, but now show how the main lemma follows from the above two.

Proof of of Lemma 9: Let Σ, ϵ_2 be as in Lemma 11. Let $c = 2 \cdot \epsilon_2$. Invoking Lemma 10 for this choice of c and Σ, let ϵ_1, Γ and T_2 be as given by Lemma 11. Now invoke Lemma 11 for this choice of Γ and let T_2 be the reduction so obtained.

We claim Lemma 9 holds for Σ, $\epsilon = \epsilon_1 \cdot \epsilon_2$ and T being the composition of T_2 with T_1.

It is clear that $T_2(T_1(\cdot))$ maps instances of Max 2-CSP-Σ to instances of the same problem. Since both T_1 and T_2 are linear time reductions, it follows that so is T. Also since both preserve satisfiability, so does their composition. Finally the unsatisfiability of $T(\phi)$ may be lower bounded as follows.

$$\text{UNSAT}(T_2(T_1(\phi))) \geq \epsilon_2 \cdot \text{UNSAT}(T_1(\phi)) \geq \min\{\epsilon_2 \cdot c \cdot \text{UNSAT}(\phi), \epsilon_2 \cdot \epsilon_1\} = \min\{2 \cdot \text{UNSAT}(\phi), \epsilon\}.$$

This concludes the proof of Lemma 9. ∎

In the following sections we comment on the proofs of Lemmas 10 and 11. Since the former is the more novel element, we defer discussion about it to the end. We start with Lemma 11.

5 Alphabet Reduction and Error Correcting Codes

In order to motivate the strength of Lemmas 11 and 10 we first describe some of the more elementary operations one can use to manipulate the parameters k and Σ. The results in the following proposition are by now either considered "basic" or at least "standard" in the context of approximation preserving reductions. The reader is strongly encouraged to think about each individually before reading the proof sketch, so as to gain some intuition into the assertions and their proofs.

Proposition 12 *Fix integers ℓ, k. There exist linear-time, satisfiability preserving reductions A_1, A_2, and A_3 such that*

1. *A_1 reduces* Max$k$-CSP-$\{0,1\}^\ell$ *to* Max$(k \cdot \ell)$-CSP-$\{0,1\}$ *with* $\text{UNSAT}(A_1(\phi)) = \text{UNSAT}(\phi)$.
2. *A_2 reduces* Max$k$-CSP-$\{0,1\}$ *to* Max3-CSP-$\{0,1\}$ *with* $\text{UNSAT}(A_2(\phi)) \geq \frac{1}{2^{k+2}}\text{UNSAT}(\phi)$.
3. *A_3 reduces* Max$k$-CSP-$\{0,1\}$ *to* Max2-CSP-$\{0,1\}^k$ *with* $\text{UNSAT}(A_2(\phi)) \geq \frac{1}{k}\text{UNSAT}(\phi)$.

Proof of Sketch: We consider the items in sequence.

1. For the first part, given a Max$k$-CSP-$\{0,1\}^\ell$ instance ϕ with constraints $C_1, \ldots, C_m$ on variables $X_1, \ldots, X_n$ taking values in $\{0,1\}^\ell$, we "encode" each variable X_i by a collection of ℓ Boolean variables $Y_{i,j}$, $j \in \{1, \ldots, \ell\}$, with the association that an assignment $\vec{a} = \langle a_1, \ldots, a_\ell \rangle \in \{0,1\}^\ell$ to X_i corresponds to the assignments $Y_{i,j} \leftarrow a_j$. A constraint $C_j = f(X_{i_1}, \ldots, X_{i_k})$ can now be naturally represented as a constraint $C'_j = f'(Y_{i_1,1}, \ldots, Y_{i_1,\ell}, \ldots, Y_{i_k,1}, \ldots, Y_{i_k,\ell})$ where f' is satisfied by an assignments to the $Y_{i,j}$'s if and only if the corresponding assignment to the X_i's satisfies f. It is easy to verify that the assignments to X_i's are in 1-to-1 correspondence with the assignments to $Y_{i,j}$'s with corresponding assignments satisfying exactly the same number of constraints. This yield the reduction A_1.

   (The important aspect to note in this reduction is that its performance degrades with ℓ. Indeed this is one of the principal effects we will aim to remedy later.)

2. For the second part, we hint that this is essentially similar to the classical reduction from "SAT" to "3SAT", whose approximability properties were clarified in [23]. In this reduction, when transforming an instance ϕ with constraints $C_1, \ldots, C_m$ on variables $X_1, \ldots, X_n$, one retains all the original variables, and adds for each constraint $C_j = f(X_{i_1}, \ldots, X_{i_k})$ a collection of $K \approx 2^k$ "auxiliary" variables $Y_{j,1}, \ldots, Y_{j,K}$ and introduce K ternary constraints $C'_{j,1}, \ldots, C'_{j,K}$ on variables $(X_{i_1}, \ldots, X_{i_k}, Y_{j,1}, \ldots, Y_{j,K})$ such that an assignment to $(X_{i_1}, \ldots, X_{i_k}) \leftarrow \vec{a}$ satisfies C_j if and only if there exists an assignment $(Y_{j,1}, \ldots, Y_{j,K}) \leftarrow \vec{b}$ such that all the constraints $C'_{j,1}, \ldots, C'_{j,K}$ are satisfied by the assignment $(\vec{a}, \vec{b})$. It can be seen that this reduction has the right properties.

3. This reduction, though also simple, is more "recent" than others, having been first brought out by the work of Fortnow et al. [15]. Here, the idea is to lump together k bit strings queried in various constraints as new single variables, but then to check their consistency against the older single bit assignments. Formally, given k-ary constraints $C_1, \ldots, C_m$ on Boolean variables $X_1, \ldots, X_n$, we create an instance with km constraints $\{C_{j,\ell}\}$ on $n + m$ variables $X'_1, \ldots, X'_n, Y_1, \ldots, Y_m$. If the constraint $C_j = f(X_{i_1}, \ldots, X_{i_k})$, then the constraint $C_{j,\ell}$ applies to variables Y_j and X'_{i_ℓ} and verifies that the k-bit assignment to $f(Y_j) = 1$, that $X'_{i_\ell} \in \{0^k, 0^{k-1}1\}$, and that the last bit of X'_{i_ℓ} equals the ℓth bit of Y_j. It is easy to see that assignments to the X' variables can be interpreted as assignments to the X variables and that the constraints $C'_{j,1}, \ldots, C'_{j,k}$ are all satisfied only if the corresponding assignment to X_i's satisfy C_j, which suffices to conclude that this reduction has the desired property.

∎

To summarize, Proposition 12 suggests a number of obvious reductions between constraint satisfaction problems. The upshot is that large gaps are hard to achieve when k and $|\Sigma|$ are small. But as it turns out the two parameters, k and $\log|\Sigma|$ are not totally similar in behavior. On the one hand, one can tradeoff Σ for a smaller alphabet, by increasing the number of queries. But reversing this tradeoff does not seem to be as obvious (and more involved results show that we do have to lose something in the unsatisfiability).

Returning to our goal of Lemma 11, of reducing a large alphabet Γ to some small fixed alphabet Σ, we see we could do this, if we were allowed to increase the number of queries (but we have to keep this fixed to 2), or allow the unsatisfiability of the reduced instance to be much smaller than (such as say $1/|\Gamma|$ times) the unsatisfiability of the source instance. But we wish to do better and lose only a fixed constant.

Turns out the prior work on PCPs, in particular [2, Section 6], addresses precisely this issue (though it was not conceived to be utilized as many times as in the current proof). The steps in the reduction resemble the classical one (Proposition 12, Part 1) however each step is significantly different.

Given an instance ϕ of max 2-CSP-Γ with constraints $C_1, \ldots, C_m$ on variables $X_1, \ldots, X_n$, we first produce an instance of max 3-CSP-$\{0,1\}$ with the following steps:

1. First we "encode" each variable X_i taking values in Γ with a collection of Boolean variables $X'_{i,1}, \ldots, X'_{i,K}$ (for some large constant K depending only on $|\Gamma|$). The classical reduction did so by representing elements of Γ as binary strings and then using the new variables to represent these binary strings (see the proof of Proposition 12, Part 1). Unfortunately this representation is not robust, and loses $1/\log|\Gamma|$ factor in the gap simply due to the fact that

two different elements of Γ may differ in only one bit in their respective binary encodings. The reduction used to prove Lemma 11 in [12] gets around this loss by representing elements of Γ in an "error-correcting code": Specifically, we find a collection S of Γ strings in $\{0,1\}^\ell$ for an appropriate integer ℓ so that every pair of strings differ in, say, at least $\ell/10$ coordinates. Such codes are well known to exist, though for our purposes it is more convenient to work with special codes.

2. Next, for a constraint, $C_j = f(X_{i_1}, X_{i_2})$, we introduce a new collection of variables $\{Y_{j,t}\}_t$ and a collection of constraints $\{C'_{j,t}\}$ on the variables $\{X'_{i_1,1}, \ldots, X'_{i_1,K}\}, \{X'_{i_2,1}, \ldots, X'_{i_2,K}\}, \{Y_{j,t}\}_t$. We won't be able to describe these constraints here, but they "enforce" two conditions: (1) They enforce that the variables $X'_{i_1,*}$, $X'_{i_2,*}$ are close encoding of some strings in the code S (e.g., changing fewer than $\ell/5$ variables $X'_{i_1,*}$ yields a string in S). (2) They enforce that the closest members of S correspond to assignments to X_{i_1} and X_{i_2} that satisfy C_j. The special aspect of the new constraints is that even though we have an enormous number of these constraints (growing with $|\Gamma|$), violating either of the conditions (1) or (2) would lead to a constant (say $3\epsilon_2$) fraction of the constraints $\{C'_{j,t}\}_t$ being violated (whereas classical reductions only violated a single constraint, when an original constraint was unsatisfied).

   Finding the right code S, the number of auxiliary variables $Y_{j,*}$ and the right collection of constraints $C'_{j,*}$ may be dismissed as a mere a "finite" search problem, if only we could prove that they exist. Unfortunately, the only proofs that we know that such structures exist, is the constructive one. And the constructive proof essentially amounts to building a "finite" PCP-like object (where failure to satisfy some conditions are "visible" to many local checks). Fortunately, these PCPs can afford to be much larger than the "polynomial sized" PCPs we seek, and their constructions are significantly simpler. Dinur presents a very compact such construction (see [12, Section 6]) while earlier constructions (e.g., [2, Section 6]) while being longer are still quite simple and natural. We remark that while the problem is a very combinatorial one, the construction of these gadgets and their analysis does rely on "algebraic" results over finite fields ([2]) or Harmonic analysis over the Boolean cube ([12]).

Once one has such a reduction from Max 2-CSP-Γ to Max3-CSP-$\{0,1\}$ one can apply a standard reduction (Proposition 12, Part 3) to now reduce the problem further to $\max$ 2-CSP-$\{0,1\}^3$ yielding Lemma 11 for $\Sigma = \{0,1\}^3$. The reader may look at [12, Section 6] for details.

6 Gap amplification

We now move to the technical centerpiece of Dinur's proof of the PCP theorem. Before getting into the specifics of this problem, we first describe the context of the result and its proof.

6.1 Background: Recycling Randomness

The underlying problem here, of amplifying gaps, plays a major role in the developing theory of "randomized computation". Since every essentially randomized algorithm errs with some positive probability, a natural question is to investigate whether this error could be reduced.

For instance, consider one of the classical (randomized) algorithms to determine if an n-bit integer is a prime. The early algorithms (cf. [22]) had the property that they would always declare prime

inputs to be "prime", but for any composite input they may declare it also to be "prime" with probability half. The classical algorithm would need an n-bit long random string to perform this test. Now, suppose we wish to reduce this error probability (of concluding that composites may be "prime") to say 1/128, one only needs to run the basic algorithm 7 times and declare a number to be prime only if every one of the seven iterations declared it to be prime. One of the drawbacks of this approach is that this process costs seven times the original cost in terms of randomness, as well as running time. While the latter may be an affordable cost (esp. for settings other than primality testing where no polynomial time deterministic algorithm is known), however, the increasing cost of randomness may prove less affordable. (Unlike the case of processor speeds in computers which under the empirically observed "Moore's Law" keep doubling every three years, physical generation of pure randomness does not seem to be getting easier over the years.) In view of this, one may ask if there is a more "randomness-efficient" way to get the error probability down to 1/128 without expending $7n$ random bits?

This task has been studied extensively under the label of "recycling randomness" [1, 11, 19] in the CS literature, which shows that it suffices to use something like $n + ck$ bits, for some absolute constant c, to reduce the error to 2^{-k} (though the cost in terms of running time remains a multiplicative factor of k). The most common technique for such "random-efficient" amplification, is to repeat the randomized algorithm with related randomness. More formally, suppose $A(x; R)$ denotes the computation of a randomized algorithm to determine some property of x (e.g., $A(x) = 1$ if and only if x is a prime integer). The standard amplification constructs a new algorithm $A'(x; R')$ where $R' = (R_1, \ldots, R_k)$ is a collection of k independent random strings from $\{0,1\}^n$ and $A'(x; R') = 1$ if and only $A(x; R_1) = \cdots = A(x; R_k) = 0$. Now, given that each invocation $A(x; R_i)$ only "leaks" one bit of information about R_i, using independent random coins is completely inessential for this process. Indeed it is easy to subsets $S \subseteq \{\{0,1\}^n\}^k$ of cardinality only $2^{(O(n+k)}$ such the performance of A' where R' is chosen uniformly from S is almost as good as when drawn from the entire universe of cardinality 2^{nk}. The computational bottleneck here is to produce such a distribution/set S efficiently.

One popular approach to producing such a set efficiently uses the technique of "random walks" on "expander graphs". Here we create a graph G whose vertices are the space of random strings of A (i.e., $\{0,1\}^n$) with the property that each vertex of G is adjacent to a fixed number, D, of other vertices in G. For the application of recycling randomness it will be important that one can enumerate in time polynomial in n all the neighbors of any given vertex $R \in \{0,1\}^n$, though for the purpose of the PCP gap amplification it will suffice to be able to compute this in time $2^{O(n)}$. The "random walk" technique to recycling randomness produces $R' = (R_1, \ldots, R_k)$ by first picking $R_1 \in \{0,1\}^n$ uniformly at random, and then picking R_2 to be a random neighbor of R_1, and R_3 to be a random neighbor of R_2 and so on. In other words R' is generated by taking a "random walk" on G.

To understand the randomness implications of this process, we first note that this process takes $n + k \log D$ bits of randomness. So it is efficient if D is small. On the other hand the amplification property relates to structural properties of the graph. For instance, the reader can see that it wouldn't help if the graph had no edges, or were just a collection of $2^n/(D+1)$ disconnected complete graphs of size $D + 1$ each! Indeed for the amplification to work well, the graph needs to be an extremely well connected graph, or an "expander" as defined next.

Definition 13 *For a graph $G = (V, E)$ and subset $S \subseteq V$, let $E_S = \{(u, v) \in E$ s.t. $|\{\mathrm{u}, \mathrm{v}\} \cap \mathrm{S}| = 1\}$ denote the set of edges crossing from S to its complement. The expansion of the set S, denoted*

$e(S)$, is the quantity $\left(\frac{|E_S|}{|E|}\right) / \left(\frac{|S|}{|V|}\right)$. G is said to be a (γ, D)-expander if every vertex is adjacent to exactly D other vertices, and every set S with $|S| \leq |V|/2$ has expansion $e(S) \geq \gamma$.

It is by now well-known in the CS literature that if R' if generated by a k-step random walk on a (γ, D)-expander, that the error probability reduces to $2^{-\delta k}$ where δ is a universal constant depending only on γ and D. (This result was first shown in a specific context by Ajtai et al. [1], and then noted for its general applicability in [11, 19].) Furthermore, a rich collection of "explicit" (γ, D)-expanders have been constructed, allowing for widespread application of this result. See [20] for a survey.

6.2 Amplification of PCPs: Naive approaches

We now return to the issue of amplifying the gap in Max 2-CSP-Σ. The naive approach to this problem would be to "iterate" the associated PCP verifier twice. The following proposition describes this operation in the CSP language.

Proposition 14 *There exists a quadratic time satisfiability preserving reduction A_4 reducing* Max 2-CSP-Σ *to* Max4-CSP-Σ *such that if* $\text{UNSAT}(\phi) = \epsilon$ *then* $\text{UNSAT}(A_4(\phi)) = 1 - (1-\epsilon)^2$.

We leave it to the reader to verify the above proposition. The main aspect to notice is that the variables of $A_4(\phi)$ are the same as the variables of ϕ, while $A_4(\phi)$ has a constraint C'_{ij} for every pair of constraints C_i, C_j of ϕ where C'_{ij} represents the conjunction of the constraints C_i and C_j.

We move on to the problems with this reduction. First, this reduction takes quadratic time. More significantly the size of the instance $|A_4(\phi)|$ is really quadratic in $|\phi|$ and this is a price we can not afford. (Logarithmically many iterations of this process would blow the instance size up from n to n^n, which completely destroys any hope of using this to construct PCPs.)

Fortunately, this is an aspect that is readily amenable to the "random walk on expanders" technique. Specifically we can consider a better k-fold amplification reduction A_5 reducing Max 2-CSP-Σ to Max($2k$)-CSP-Σ as follows: The variables of $A_5(\phi)$ are the same as the variables of ϕ. Constraints of $A_5(\phi)$ are generated by first picking k constraints of ϕ by performing a k-step random walk on a (γ, D)-expander G with m vertices (so the vertices of G correspond to constraints of ϕ) and then taking the conjunction of all such constraints. The number of constraints now is only $n \cdot D^k$ which is linear in n if k, D are constant. The analysis used in the general setting of recycling randomness can now be used to prove the following proposition.

Proposition 15 *There exists a constant $\delta > 0$ such that for every k, there exists a linear time satisfiability preserving reduction A_5 reducing* Max 2-CSP-Σ *to* Max($2k$) $-$ -CSP-Σ *such that if* $\text{UNSAT}(\phi) = \epsilon$ *then* $\text{UNSAT}(A_4(\phi)) = 1 - (1-\epsilon)^{\delta k}$.

The amplification effects of the above proposition, as well as the time complexity are now as we would like. However there is still one, fatal, flaw with both reductions above. They do not reduce "binary" constraint satisfaction problems to "binary" constraint satisfaction problems. Instead they reduce them to $(2k)$-ary constraint satisfaction problems, which is also of no use in the iterative approach. So we turn to the problem of preserving the "binary" nature of constraints.

6.3 Background: Parallel Repetition

For this section it is convenient to switch to the PCP language. Consider a PCP verifier V that on input x and random string R two queries $q_1(R)$ and $q_2(R)$ to an oracle $\Pi : \mathbb{Z}^+ \to \Sigma$ and accepts if the responses $a = \Pi(q_1(R))$ and $b = \Pi(q_2(R))$ satisfy $f(R, a, b) = 1$ for some fixed predicate f depending on x.

The naive amplification (corresponding to reduction A_4 described earlier) corresponds to the following verifier V': V' picks two random strings R_1, R_2 from the space of the randomness of V and issues queries $q_1(R_1), q_2(R_1), q_1(R_2), q_2(R_2)$ to Π. If the responses are a_1, b_2, a_2, b_2 then V' accepts if $f(R_1, a_1, b_1) = 1$ and $f(R_2, a_2, b_2) = 1$. The acceptance probability of the modified verifier V' (maximized over Π) is the square of the acceptance probability of V (maximized over Π), which is good enough for us. However it makes 4 queries and this is the issue we wish to address in this section.

One natural attempt at reducing the number of queries may be to "combine" queries in some natural way. This is referred to as parallel repetition of PCPs. In the k-fold parallel repetition we consider an new verifier $V^{||\otimes k}$ that accesses an oracle $\Pi^{||\otimes k} : (\mathbb{Z}^+)^k \to \Sigma^k$ (with the association that the k coordinates in the domain correspond to k queries to Π, and the k coordinates in the range to the k responses of Π) and functions as follows: $V^{||\otimes k}$ picks k independent random strings $R_1, \ldots, R_k$ and queries $\Pi^{||\otimes k}$ with $(q_1(R_1), \ldots, q_1(R_k))$ and $(q_2(R_1), \ldots, q_2(R_2))$. If the responses of $\Pi^{||\otimes k}$ are $(a_1, \ldots, a_k)$ and $(b_1, \ldots, b_k)$ then $V^{||\otimes k}$ accepts if $f(R_i, a_i, b_i) = 1$ for every $i \in \{1, \ldots, k\}$.

One may hope that the error in the k-fold parallel repetition goes down exponentially with k. However, any such hopes are dashed by the following example, which gives a choice of (Σ, f, q_1, q_2) such that the error of the k-fold parallel repetition *increases* exponentially with k.

Example: Let V work with $\Sigma = \{0, 1\}$ and the space of random strings R be $\{0, 1\}$. Let $q_i(R) = i + R$ and let $f(0, a, b) = b$, and $f(1, a, b) = 1 - a$. The reader may verify that for every oracle $\Pi : \{1, 2, 3\} \to \{0, 1\}$ the acceptance probability of V is $\frac{1}{2}$. Furthermore there exist $\Pi^{||\otimes k}$ for which the acceptance probability of $V^{||\otimes k}$ is $1 - 2^{-k}$.

The example illustrates some of the many problems with naive hopes one may have from parallel repetition. In the face of the above example one may wonder if any amplification is possible at all in this setting. After many works exploring many aspects of this problem, Raz [24] gave a dramatic *positive*. He considers restricted verifiers whose "question" spaces (the image of $q_1(\cdot)$ and $q_2(\cdot)$) are disjoint, and shows that for such verifiers, error does reduce exponentially with the number of iterations, with the base of the exponent depending only on the acceptance probability of the original verifier, and the answer size $|\Sigma|$. Furthermore, there exist reductions reducing any verifier to a restricted verifier only a constant factor in the gap. (The reader may try to see how one such reduction is implied by Proposition 12, Part 3.) Combined these two steps allow us to amplify the gap in PCPs — but now we have lost the "linear time property".

Is it possible to try parallel repetition while recycling randomness? Given the difficulty in analyzing parallel repetition (Raz's proof, while essentially elementary, is already one of the most intricate proofs seen in the PCP setting) the task of combining it with recycling randomness appears forbidding. Remarkably enough Dinur [12] manages to combine the two techniques and achieve the desired gap amplification, and does so with relatively simple proofs. Among other things, Dinur's realization is that even an example such as the above may not defeat the purpose. For the purposes of Lemma 10 it suffices to show that the acceptance probability goes down, provided it was very high to start with; and that in the remaining cases it remains bounded away from 1 (by say, 2^k).

Dealing with cases where the acceptance probability is very high (e.g., greater than $1 - \Sigma^{-k}$) turns out be easier than dealing with the other cases. We now describe Dinur's gap amplification.

6.4 Gap amplification

To describe Dinur's gap amplification lemma we switch back to the terminology of CSPs. Her main idea is to consider the graph underlying a Max 2-CSP-Σ instance, and to impose some structure on the graph on it, and then to generate instances of Max2-CSP-Σ^K based on walks of length k on this graph.

We start by describing the graph G_ϕ underlying a $\max 2csps$ instance ϕ. G_ϕ has n vertices corresponding to the n variables of ϕ and (i, j) is an edge if there is some constraint (among the m constraints of ϕ) on the pair of variables X_i, X_j. (As an aside, Dinur's analysis relies essentially on the feature that this graph is "undirected" i.e., $(i, j) \in E \Leftrightarrow (j, i) \in E$, which is in sharp contrast to Raz's setting which requires that $(i, j) \in E$ implies that $(i, i') \notin E$ and $(j, j') \notin E$ for any i', j'.)

As a first step, Dinur performs some preprocessing to ensure that G_ϕ is a (γ, D)-expander. If it is not, she reduces, in linear time, the instance ϕ to a different instance $\tilde{\phi}$ of Max 2-CSP-Σ so that $\mathrm{UNSAT}(\tilde{\phi}) \geq \epsilon_3 \cdot \mathrm{UNSAT}(\phi)$. This preprocessing reduction (from ϕ to $\tilde{\phi}$) is achieved by first transforming ϕ to ϕ_1 so that G_{ϕ_1} has bounded degree, which also uses expanders in a technique going back to the work of [23]. Next it transforms ϕ_1 to $\tilde{\phi}$ by imposing a collection of vacuous constraints ϕ_2 (which are always satisfied) such that G_{ϕ_2} is an expander. It may be verified that if G_{ϕ_2} is a $(2\gamma, D/2)$-expander and G_{ϕ_1} has degree $D/2$, then the union of the two graphs yields a (γ, D) expander. If one can amplify the gap of the instance $\tilde{\phi}$ by c/ϵ_3 factor, then the composition of the two steps amplifies the gap of ϕ by a factor c. This (to simplify our notation) below we assume that G_ϕ is an expander.

We now move to the crux of Dinur's amplification. Given ϕ as above, let k correspond to the number of repetitions we intend to attempt. For $u \in V(G_\phi)$, let $B(u, k)$ denote the set of vertices within a distance of at most k from u. Let $K = \max_u\{|B(u, k)|\} \leq \sum_{i=0}^{k} D^i$. Then the new alphabet $\Gamma = \Sigma^K$. The new instance ϕ' will continue to have n variables (same as ϕ), where the new variable X'_u will be viewed as assigning an opinion on its value of the assignment to X_v for every v that is within a distance of k from u in G_ϕ. (Notice that the number of such v's for any fixed u is at most K and so indeed an alphabet of size $|\Sigma|^K$ suffices to represent all these opinions.) We use $X'_u(v)$ to denote the opinion of u about v.

Now for the constraints of ϕ': For every walk w in G_ϕ starting at vertex u and ending at v of length $\ell \in [k/2, k]$ ϕ' has $D^{k-\ell}$ *copies* of the constraint $F(X'_u, X'_v)$ which imposes the conjunction of all constraints within balls of radius k of u and v. Specifically, (1) For every $u' \in B(u, k) \cap B(v, k)$ it must hold that $X'_u(u') = X'_v(u')$. (2) For $u' \in B(u, k) \cup B(v, k)$ let $O(u) = X_u(u')$ or $X_v(u')$ whichever is defined (notice by (1) that these are consistent). For every u', v' such that $u', v' \in B(u, k) \cup B(v, k)$ and $f(u', v')$ is a constraint of ϕ, it must hold that $f(O(u'), O(v')) = 1$. Notice that the many *copies* of each constraint ensure that a randomly chosen constraint of ϕ' will correspond to a walk whose length is distributed uniformly over the interval $k/2, \ldots, k$.

The above gives the complete description of the reduction essentially used in Dinur's work. We won't be able to give the proof as to why it works here. Even worse, we won't even be able to motivate the reasons behind the many delicate choices made in the reduction above. (Why are the new variables chosen as they are? Why do we create walks of so many different lengths? Why do we replicate the constraints in this way?) All we can say is that these choices are not necessarily

the first ones one may consider, but definitely make the proof of the amplification lemma very easy. The reader is encouraged to followup by reading the original paper.

7 Conclusion

We hope the reader finds the above description to be somewhat useful, and motivating when reading Dinur's new approach to consruction of PCPs. We remark that the earlier algebraic approaches, while technically much more complicated, do have some appealing high level views. The reader is pointed to the work of Ben-Sasson and this author [10] to get a sense of some of the work in the older stream.

Moving on beyond the specific proofs, and constructions used to get probabilistically checkable proofs, we hope that the notion itself is appealing to the reader. The seemingly counterintuitive properties of probabilistically checkable proofs highlight the fact the "format" in which a proof is expected is a very powerful tool to aid the person who is verifying proofs. Indeed for many computer generated proofs of mathematical theorems, this notion may ease verifiability, though in order to do so, PCPs need to get shorter than they are; and they verification scheme simpler than it is. Dinur's work helps in this setting, but much more needs to be done.

And finally, moving beyond the notion of proofs, we also hope this article reminds the reader once more of a fundamental question in logic, and computation, and indeed for all mathematics: Is P=NP? Can we really replace every mathematician by a computer? If not, would it not be nice to have a proof of this fact?

References

[1] M. Ajtai, J. Komlos, and E. Szemeredi. Deterministic simulation in logspace. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 132–140, 1987.

[2] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[3] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.

[4] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd ACM Symposium on the Theory of Computing*, pages 21–32. ACM, New York, 1991.

[5] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[6] László Babai and Shlomo Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity class. *Journal of Computer and System Sciences*, 36(2):254–276, April 1988.

[7] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCP's and non-approximability — towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

[8] Mihir Bellare, Shafi Goldwasser, Carsten Lund, and Alex Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pages 294–304. ACM, New York, 1993.

[9] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 113–131, 1988.

[10] Eli Ben-Sasson and Madhu Sudan. Short PCPs with poly-log rate and query complexity. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 266–275, 2005.

[11] Aviad Cohen and Avi Wigderson. Dispersers, deterministic amplification, and weak random sources (extended abstract). In *IEEE Symposium on Foundations of Computer Science*, pages 14–19, 1989.

[12] Irit Dinur. The PCP theorem by gap amplification. Technical Report TR05-046, ECCC, 2005. Revision 1, Available from `http://eccc.uni-trier.de/eccc-reports/2005/TR05-046/`.

[13] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP-theorem. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 155–164, 2004.

[14] Uriel Feige, Shafi Goldwasser, Laszlo Lovasz, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996.

[15] Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.

[16] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[17] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[18] Johan Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001.

[19] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *IEEE Symposium on Foundations of Computer Science*, pages 248–253, 1989.

[20] Nati Linial and Avi Wigderson. Expander graphs and their applications: Lecture notes of a course given at the Hebrew University, 2003. Available from `http://www.math.ias.edu/~avi/TALKS/expander_course.ps`.

[21] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, September 1994.

[22] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[23] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

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

[25] Madhu Sudan. PCP and inapproximability: Survey and open problems, February 2000. Slides of Talk given at DIMACS Workshop on Approximability of NP-hard problems, Nassau Inn, Princeton, NJ. Available from `http://theory.csail.mit.edu/~madhu/slides/dimacs00.ps`.