On Dinur's Proof of the PCP Theorem

Jaikumar Radhakrishnan[*] Madhu Sudan[†]

August 25, 2006

Abstract

Probabilistically checkable proofs are proofs that can checked probabilistically by reading very few bits of the proof. In the early 1990's, it was shown that proofs could be transformed into probabilistically checkable ones with only a modest increase in their size. The initial transformations, though elementary, were a little too complex. A recent work due to Irit Dinur gives a dramatically simple (and radically new) construction of probabilistically checkable proofs. This article explains the notion of a probabilistically checkable proof, presents the formal definition and then introduces the reader to Dinur's work along with some of the context.

1 Introduction

As advances in mathematics continue at the current rate, editors of mathematical journals increasingly face the challenge of reviewing 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. One of the main aims 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).

[*]Tata Institute of Fundamental Research, Mumbai, India & Toyota Technological Institute at Chicago (TTI-Chicago), University Press Building, 1427 East 60th Street, Chicago, IL 60637. jaikumar@tti-c.org.

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

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).

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; the verifier 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 specific 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, any such given format 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 differs 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 values from the set $\{0, 1\}$ equiprobably); 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: What is the tradeoff between the query complexity and the error incurred by the verifier? It must be stressed at this point that we require the error probability to be 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.)

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 [24, 10, 14, 23, 8, 22, 7], 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 need to be queried? The first sequence of results [9, 8, 22] 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, [8] obtained nearly linear sized PCPs.) The second short sequence [7, 6] 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 [6] 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 [13, 12] attempted to reduce this constant and culminated in yet another surprise. Håstad [26] 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 focused 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 [19].)

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., [35, Page 12] for a look at the ingredients needed to get the PCP theorem of [26]). Recently, Dinur and Reingold [20] 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 [19] 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 much of this article. 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 setting up some basic notation and background in computational complexity in Section 2. We then introduce the formal definition of a PCP and a statement of the main result in Section 3. In this section we also introduce an alternate view of PCPs as forms of optimization problems. This language is useful in explaining Dinur's approach to constructing PCPs. In Section 4 we then describe Dinur's approach at a high-level 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 probabilistic verifier of proofs. In Sections 6 and 5 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 Preliminaries

We start by fixing some basic notation (and introducing some basic concepts such as strings and graphs).

In the sequel $\mathbb{R}$ 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 $E : D \to \{\text{true}, \text{false}\}$, 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 terminology from "graph theory". Formally a graph G is given by a pair (V, E). The set V is finite and its elements are called vertices. The set E is the set of edges. Each edge $e \in E$ will a be associated with two (not necessarily distinct) vertices, its head and tail, denoted by $\mathsf{head}(e)$ and $\mathsf{tail}(e)$; We say that e is of the form (u, v) if $\mathsf{tail}(e) = u$ and $\mathsf{head}(e) = v$; we then say that e leaves u and enters v. The edge sets for our graphs will be symmetric, that is, each edge e of the for (u, v) will be paired with a unique edge of the form (v, u); this edge is is called e^-. If e is a self-loop, that is, for the form (u, u), then $e^{-1} = e$. If $(u, v) \in E$, then we refer to v as being adjacent to u, or being a neighbor of u. The number of edges leaving u is called the degree of u and denoted by $\deg(v)$ (a self-loop on v contributes one to the degree, not two). We say a graph d-regular if every vertex has degree d; thus, for a d-regular graph (V, E), we have $|E| = d|V|$. 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 will also need in Section 6 a common extension of the notion of graphs named hypergraphs. As in graphs, hypergraphs are described by a collection of vertices and (hyper)edges. with the novelty that the hyperedges may form relations on more than two vertices. Formally, for a positive integer r, an r-uniform hypergraph is described by a pair (V, E) where V is a finite set and E is a set of subsets of V, with every $e \in E$ being of size r.

2.1 Computational complexity: P, NP, Proofs and Optimization

The theory of computation is broadly concerned with the task of computing functions, or solving "search" problems. A functional computational problem is described by a function $f : \{0,1\}^* \to \{0,1\}^*$ and the goal is to compute $f(x)$, given x. The running time of an algorithm A computing f is measured as a function of the input size and denoted $T_A(n)$, where $T_A(n)$ is the maximum over all inputs $x \in \{0,1\}^n$ of the running time of A on x. If there is a polynomial $p(n)$ satisfying $T_A(n) \leq p(n)$ for every n, then A is said to be a polynomial time algorithm. The class of all Boolean functions $f : \{0,1\}^* \to \{0,1\}$ that are computable by polynomial time algorithms denotes the class P. Note that computing a Boolean function f is equivalent to deciding membership in the *language* (computer scientist's phrase for subsets of $\{0,1\}^*$) $L_f = \{x : f(x) = 1\}$.

Many computational problems are not quite functional, but more about solving "search" problems. Consider for example the classical graph coloring problem where the input is a graph $G = (V, E)$ and an integer k and the goal is to find a coloring $A : V \to \{1, \ldots, k\}$ such that for every edge e of the form (u, v), it is the case that $A(u) \neq A(v)$, if such a coloring exists. Problems such as coloring are characterized by the fact that there may be many "solutions" to the problem and the goal is to find any one (or simply determine if a solution exists). Formally, such problems may be described by a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$. Given an input $x \in \{0,1\}^*$ the goal is to find a solution $y \in \{0,1\}^*$ such that $(x, y) \in R$. (In the case of graph coloring, the input x is the pair (G, k), and the solution y is is the coloring A.) For simplicity we often focus on the (slightly) simpler problem of merely deciding whether x has a solution y (such that $(x, y) \in R$). This defines a language associated with a relation R, $L_R = \{x : \exists y \text{ s.t. } (x, y) \in R\}$.

The class NP consists of all languages L_R where R has "polynomially short solutions" (i.e., there is a polynomial q such that for every pair $(x, y) \in R$ it is the case that $|y| \leq q(|x|)$) and for which deciding validity of a solution (i.e., deciding whether $(x, y) \in R$) is in P. Graph coloring is a common example of a problem in NP. Other well-known ones include the Traveling Salesman Problem, Bin Packing, Finding large cliques in graphs, deciding satisfiability of a logical formula etc.

In addition to being in the class NP, the above problems also share the common feature of being notoriously hard. No polynomial time algorithm is known to solve these problems to date. To explain this commonality (and to justify somewhat the common difficulty) the theory of NP-completeness was developed in the early 70s by Cook [18], Levin [30], and Karp [29]. Informally, an NP language L is said to be NP-complete if it is provably harder than every other problem in NP. One way to formalize this is to say that for every NP language L' there exists a polynomial time computable transformation T such that for every $x \in \{0,1\}^*$ $x \in L'$ if and only if $T(x) \in L$. It turns out that all the problems mentioned in the previous paragraph are also NP-complete. So in fact they are equivalent to each other, explaining their commonality. It is a widely held belief that P $\neq$ NP, though at this point this remains a conjecture.

One particular NP-complete problem is the following. Fix a (reasonable) system of logic whose theorems and proofs are written as binary strings. Then the language SHORT PROOFS that consists of all theorems T that have proofs of length at most, say, $|T|^2$ is an NP-complete language. This is true since in any reasonable system of logic, it should be "easy" to verify proofs, where the informal term "easy" can be formalized by the requirement that the proof can be verified in time polynomial in its length. Part of the argument in favor of the belief that P does not equal NP stems from the language SHORT PROOFS. If P were equal to NP there would essentially be an "automated" efficient way to prove theorems — something that does not seem feasible. Indeed it was this perspective, the study of the complexity of theorem proving, that led Cook [18] to define the class NP.

On the other hand, much of the interest in NP-completeness arises from the perspective it sheds on "combinatorial optimization", the field that looks for efficient methods to solve optimization problems. Many of the above mentioned problems including graph coloring, bin packing, and the traveling salesman problem are instances of optimization problems. NP-completeness thus provides a bridge between such combinatorial optimization and logic by showing that many optimization problems are as hard to solve as proving theorems. In what follows we will formally describe the notion of probabilistically checkable proofs, and show how it adds to the study of optimization.

3 Probabilistically checkable proofs

A proof system is formalized by its "verifier", i.e., the algorithm that, given an assertion and supporting argument, verifies if the argument proves the assertion. Formally such a verifier is given by (an algorithm to compute) a function $V : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. that $V(T, \Pi) = 1$ implies that the assertion T is a theorem with Π being a proof.

As mentioned earlier, we are going to enhance classical verifiers/algorithms by endowing them with access to random strings and oracles. We will denote random strings just like other strings. An 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. The output 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 3.1 *For functions $r, q, a : \mathbb{Z}^+ \to \mathbb{Z}^+$ 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 3.2 *For positive constant $0 \le s \le 1$, we say that an (r,q,a)-restricted PCP verifier V accepts a language $L \subseteq \{0,1\}^*$ with 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 for every R $V^\Pi(x : R) = 1$.*

Soundness: *If $x \notin L$ then for every $\Pi : \mathbb{Z}^+ \to \{0,1\}^{a(n)}$ it is the case that $\Pr_R[V^\Pi(x : R) = 1] \le s$.*

By $\mathrm{PCP}_s[r,q,a]$ we denote the class of all languages L such that there exists an (r,q,a) restricted PCP verifier accepting L with 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.

The early results [9, 8, 22] could be described as showing that there exist polynomials $p_1, p_2 : \mathbb{Z}^+ \to \mathbb{Z}^+$ such that $\mathrm{NP} \subseteq \mathrm{PCP}_{1/2}[p_1(\log n), p_2(\log n), 1]$. The PCP Theorem, whose new proof we hope to outline later, may now be stated formally as.

Theorem 3.3 ([7, 6]) *There exist a constant q such that* $NP = \bigcup_{c \in \mathbb{Z}^+} PCP_{\frac{1}{2}}[c \log n, q, 1]$.

Finally, the state of the art result along these lines is that of Håstad [26] (see also [25]), which shows that for every $\epsilon > 0$, $\mathrm{NP} = \cup_{c\in\mathbb{Z}^+}\mathrm{PCP}_{\frac{1}{2}+\epsilon}[c \log n, 3, 1]$.

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

3.1 PCPs and Approximations

One of the principal weaknesses in the classical applications of the theory of NP-completeness to optimization problems was its inability to make any statements about the complexity of finding nearly optimal solutions to optimization problems. The PCP theorem provided a means to address this issue. Below we explain the issues in a little more detail in the context of graph coloring.

Consider the following optimization problem related to graph coloring. Given a graph $G = (V, E)$ and integer k, find a coloring $A : V \to \{1, \ldots, k\}$ that maximizes the number of "satisfied" edges, where an edge $e = \{u, v\}$ is said to be satisfied if $A(u) \neq A(v)$. If $\mathrm{P} \neq \mathrm{NP}$ then it is clear that this maximization problem cannot be solved in polynomial time (since such an algorithm could be used to determine in polynomial time if G is k-colorable, which is NP-complete). However, is it hard to compute a coloring A that is guaranteed to satisfy many edges, say at least $.999M$ edges, where M denotes the number of edges that are satisfied by the optimal coloring? Such a coloring would be termed a .999 approximation to to the coloring problem, and is often a very good substitute for the "optimal" coloring. Significant research thus far has failed to yield a polynomial time .999 approximation algorithm for the coloring problem, but yet the theory of NP-completeness did not immediately rule out such an algorithm. All it rules out are algorithms that satisfy M edges, but an algorithm that satisfies $M - 1$ edges might still be conceivable.

To extend the theory of NP-completeness to rule out a .999 approximation to graph coloring it would be useful to have a transformation T from some NP-complete language L to graph k-coloring (for some fixed k) with the following properties:

- If $x \in L$ then $T(x)$ is k-colorable.
- If $x \notin L$ then no k-coloring satisfies .999 fraction of the edges of $T(x)$.

Such a transformation would then rule out a polynomial time .999 approximation algorithm for graph k-coloring, assuming $\mathrm{P} \neq \mathrm{NP}$ (since then we could use the algorithm's output on $T(x)$ to decide if $x \in L$ or not).

The PCP theorem it turns out does indeed produce such transformations (and rules out α-approximation algorithm for graph k-coloring for some $\alpha < 1$ though this α may be larger than .999.) We won't give such a transformation for the graph k-coloring problem, but a generalization of this problem. This generalization, which we call constraint graph coloring, turns out to be central to Dinur's proof of the PCP theorem. As in graph coloring, here also the task is to find a k-coloring of a given graph that maximizes the number of satisfied edges. However we allow more general (more restrictive) conditions for satisfiability of edges. In fact the exact conditions are part of the input description. We present this definition formally below.

Definition 3.4 (Constraint graph, coloring)

Constraint Graph: *A* constraint graph *G is a tuple $\langle V, E, \Sigma, \mathcal{C}\rangle$, where (V, E) is an undirected graph (we allow multiple edges and self-loops), Σ is a finite set called the alphabet of G, and $\mathcal{C}$ is a collection of constraints, $\langle c_e : e \in E\rangle$, where each c_e is a function from $\Sigma \times \Sigma$ to $\{0, 1\}$.*

Coloring and Satisfaction: *A* coloring *of G is a function $A : V \rightarrow \Sigma$. We say that the coloring A satisfies an edge $e = \{u, v\}$, if $c_e(A(u), A(v)) = 1$. We say that the coloring A satisfies G, if A satisfies all edges in G. If there is a coloring that satisfies G, then we say that G is satisfiable. We say that G is ϵ-far from satisfiable if every coloring leaves at least a fraction ϵ of the edges of G unsatisfied. Let*

$$\textsc{unsat}(G) = \max\{\epsilon : G \textit{ is } \epsilon\textit{-unsatisfiable}\} = \min_{A} \frac{|\{e : A \textit{ does not satisfy } e\}|}{|E|}.$$

We use $\mathcal{G}_K$ to denote the set of constraint graphs with an alphabet of size K.

Remark: The constraints in the above definition are allowed to be arbitrary functions of two inputs, each taking values in the set Σ.

As with other optimization problems, we have a natural language associated with constraint graphs, where given a constraint graph we are required to determine if it is satisfiable. We will refer to this as the *constraint graph coloring problem.* Since the constraint graph coloring problem generalizes graph k-coloring, it is also NP-complete. Dinur proves the following considerable strengthening of this assertion that says that the existence of an efficient algorithm that approximates $\textsc{unsat}$ sufficiently closely would imply that P = NP.

Theorem 3.5 (Main Theorem) *There is a constant $\epsilon_0 > 0$, such that for every language L in NP, there is a polynomial-time transformation T mapping input instances of L to $\mathcal{G}_{16}$, such that*

- *if $x \in L$, then $T(x)$ is satisfiable;*
- *if $x \notin L$, then $T(x)$ is ϵ_0-far from satisfiable.*

We now show that the above theorem is closely related to the PCP theorem.

Proposition 3.6 *The following are equivalent.*

1. *There is a constant $\epsilon_0 > 0$, such that for for every language L in NP, there is a polynomial-time transformation T mapping input instances of L to $\mathcal{G}_{16}$, such that*
   - *if $x \in L$, then $T(x)$ is satisfiable;*
   - *if $x \notin L$, then $T(x)$ is ϵ_0-far from satisfiable.*
2. *There is a constant $\epsilon_0 > 0$ such that NP $\subseteq \mathsf{PCP}_{1-\epsilon_0}[r, 2, 2]$, where $r = O(\log n)$.*

Proof:

(1) $\Rightarrow$ (2) Fix an input x, and consider the constraint graph $T(x)$. The proof oracle is the coloring $\Pi : V(T(x)) \to \{0,1\}^4$. We may assume that the number of edges in $T(x)$ is a power of two. The verifier tosses her coins and based on the outcome picks a random edge e of the form (u, v) (with constraint c_e) of $T(x)$, reads $\Pi(u)$ and $\Pi(v)$ from the proof, and accepts if and only the constraint $c_e(\Pi(u), \Pi(v)) = 1$.

(2) $\Rightarrow$ (1) Fix an input x. Consider the oracle Π. We construct a constraint graph G with one vertex of each location of Π that could be probed by the verifier. Thus, there are at most $2^{r(|x|)}$ vertices in G; since $r = O(\log n)$, there are only polynomially many vertices. For each random string $R \in \{0,1\}^{r(|x|)}$ we introduce an edge e_R. For this random string R, if the verifier queries the oracle at locations u and v, then the edge connects the vertices u and v of G, and the constraint evaluates to 1 if and only if the values assigned to u and v cause the verifier to accept.

■

Syntactically the proposition does not prove the equivalence between the PCP theorem (Theorem 3.3) and the Main theorem (Theorem 3.5). But it is straightforward to see that $\mathsf{PCP}_{1-\epsilon_0}[r, 2, 4] \subseteq \mathsf{PCP}_{1-\epsilon_0}[r, 8, 1] \subseteq \mathsf{PCP}_{(1-\epsilon_0)^c}[r \cdot c, 8 \cdot c, 1]$ for every $c \in \mathbb{Z}^+$ and so the Main theorem does imply the PCP theorem. The other direction, showing $\mathsf{PCP}_{1/2}[r, q, 1] \subseteq \mathsf{PCP}_{1-\epsilon_0}[c{\cdot}r, 2, 2]$ for some $c < \infty$ and $\epsilon_0 > 0$, is also not too hard to show, and Proposition 6.2 essentially shows such a reduction. Modulo this caveat, the proposition above does show how the PCP theorem shows the inapproximability (to within a factor of $1 - \epsilon_0$) of the constraint graph coloring problem. Once one has such a hardness, one can then use transformations to show the hardness of approximating many other optimization problems. We won't do so here, but refer the reader to [5] for a collection of such results.

We now move to the task of describing Dinur's proof of the Main 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. Though the prior approaches to proving the PCP theorem were typically stated in the "$\mathrm{PCP}_s[r, q, a]$" notation, the effective equivalence with constraint graph coloring allows us to interpret them in the latter language and we do so here.

4.1 Previous approaches

One of the principal issues to focus on is the "Gap" in the unsatisfiability achieved by the transformation. Notice that the reduction we seek creates a gap between the unsatisfiability of the instances when $x \in L$ and the unsatisfiability when $x \notin L$; the former is 0 but the latter is at least some absolute constant ϵ. We refer to this quantity as the "gap" achieved by the reduction.

The 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 constraint graph K-coloring for finite $K = |\Sigma|$ with a positive gap, the prior approaches essentially allowed K to grow with $n = |x|$. The results of Babai et al. [9, 8] and Feige et al. [22] 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 constraint graph $K(n)$-coloring for $K(n) \approx 2^{(\log n)^c}$ for some constant c.[1] Arora and Safra [7] showed that transformations implicit in the PCP constructions could be composed in a non-obvious way to reduce the alphabet size dramatically. For instance, their technique could reduce $K(n)$ to $\log(\log(\cdots(\log n)))$ for any finite number of logarithms! This motivated the search for new PCPs that would compose better. Arora et al. [6] produced two such PCPs whose composition (several times) led to $K(n) = O(1)$. In both the above cases, the composition reduced the gap by a constant factor, but this was okay since the composition was only applied a finite number of times.

Thus the previous approach could be described as constructing PCPs by "alphabet reduction", subject to "gap preservation". In contrast, Dinur's approach does quite the opposite. It starts with a reduction from the NP complete language L to Max k-CSP-Σ which has minimal gap (producing only $\mathrm{UNSAT}(\phi) \geq 1/\mathrm{poly}(n)$ 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".

4.2 Dinur's approach

The starting point for Dinur's transformation is the NP-completeness of graph 3-coloring problem, which we now state in a form suitable to us.

[1]Strictly speaking, the early results also did not achieve a constant gap in our notation, but we will ignore this issue to make the explanation simple.

Theorem 4.1 *For every language L in NP, there is a polynomial-time transformation T_0 mapping input instances of L to $\mathcal{G}_{16}$, such that*

- *if $x \in L$, then $T_0(x)$ is satisfiable;*
- *if $x \notin L$, then $T_0(x)$ is $(\frac{1}{m})$-far from satisfiable, where m is the number of constraints in $T_0(x)$.*

The overall strategy for proving Theorem 3.5 is as follows. We invoke Theorem 4.1 and obtain a transformation that takes the input instances of the given language L to constraint graphs (in fact, instances of the graph 3-coloring problem) with an inverse polynomial gap. Next, we work towards increasing this gap by repeatedly applying a specially designed transformation. Each application of this transformation at least doubles the gap, while increasing the size of the graph by a constant factor. After roughly $O(\log n)$ applications (where n is the size of the input instance for L) of the transformation, we obtain the required constraint graph. The final transformation that justifies Theorem 3.5 has the form $x \mapsto T^{(\ell)}(T_0(x))$, where $\ell = O(\log |x|)$, T_0 is transformation promised by Theorem 4.1, and T is a special gap-amplifying transformation invented by Dinur [19] which we will get to shortly.

First, we will work in this article with a slightly restricted class of constraint graph instances. These restrictions are not essential to our lemmas, however they do simplify the proof of one of them (Lemma 4.6).

Definition 4.2 (Restricted constraint graphs) *An instance $G \in \mathcal{G}_K$ is said to be* restricted *if $K = 2^k$ for some integer k, the alphabet $\Sigma = \mathbb{F}_2^k$ is the vector space of k-dimensional vectors over $\mathbb{F}_2$ the finite field of two elements, and every constraint $c_e(A(u), A(v))$ is given by a collection of degree two polynomials $P_1, \ldots, P_m$ over $2k$ variables from $\mathbb{F}_2$ and the constraint c_e is satisfied if and only if for every polynomial P_j we have $P_j(A(u), A(v)) = 0$. (Note that since each color, $A(u)$ as well as $A(v)$, is a k-dimensional vector over $\mathbb{F}_2$, together their coordinates give a $2k$-dimensional vector over $\mathbb{F}_2$, at which P_j can be evaluated.) We use $\tilde{\mathcal{G}}'_K$ to denote the set of restricted constraint graphs with alphabet of size K.*

We need one more definition before we can describe the properties of Dinur's transformation T.

Definition 4.3 (Linear-size transformation) *Let $\mathcal{G}'$ and $\mathcal{G}''$ be sets of constraint graphs. A polynomial-time computable function $T : \mathcal{G}' \to \mathcal{G}''$ is said to be an (α, ϵ)-transformation if*

1. *for all $G \in \mathcal{G}'$, if G is satisfiable then $T(G)$ is satisfiable; and*
2. *for all $G \in \mathcal{G}'$, $\text{UNSAT}(T(G)) \geq \min\{\alpha \cdot \text{UNSAT}(G), \epsilon\}$; and*
3. *the size of the output is bounded by a linear function of the size of the input, that is, there exist constants $a, b \geq 0$ such that for all $G \in \mathcal{G}'$, $|T(G)| \leq a|G| + b$.*

Using this notation, we can now present the gap amplifying transformation referred to above.

Lemma 4.4 (Main Lemma) *There is a constant $\epsilon > 0$, such that there exists a $(2, \epsilon)$-transformation from $\tilde{\mathcal{G}}_{16}$ to $\tilde{\mathcal{G}}_{16}$.*

Dinur actually works directly with unrestricted instances. In this article we restrict ourselves to the restricted version; as we see below this is sufficient for our purposes.

We remark that 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 [19, Section 7]). Later in this section, we describe the main ingredients in the proof of Lemma 4.4; the detailed proof appears in Section 5. Before we proceed to these, let us formally show that Theorem 3.5 follows immediately from Theorem 4.1 and Lemma 4.4.

Proof of Theorem 3.5: Let L be a language in NP, and let x be an input instance of L. We start by creating a transformation from L to T_0 which satisfies the property that $T_0(x) \in \tilde{\mathcal{G}}_{16}$ and $T_0(x)$ is satisfiable if and only if $x \in L$. Given $x \in \{0,1\}^n$, the transform T_0 first produces a a graph G such that G is 3-colorable if and only if $x \in L$, using the polynomial time transformation promised in Theorem 4.1. It then transforms G into an instance of $\tilde{\mathcal{G}}_{16}$ as follows: The set of vertices and edges of the constraint graph is the same as that of G; and the alphabet is $\{0,1\}^4$. The three colors $\{0,1,2\}$ are encoded using the binary strings $\{0000, 0100, 1000\}$. For each edge $\{u, v\}$ of G, we have the system of polynomial equations

$$\begin{aligned}
A(u)_1, A(u)_2 &= 0; \\
A(v)_1, A(v)_2 &= 0; \\
A(u)_3 &= 0; \\
A(u)_4 &= 0; \\
A(v)_3 &= 0; \\
A(v)_4 &= 0; \\
(A(u)_1 - A(v)_1 + 1) \cdot (A(u)_2 - A(v)_2 + 1) &= 0 \pmod 2.
\end{aligned}$$

The first six equations ensure that only valid colors are used to color u and v; the last equation ensures that the colors assigned to u and v are different. This final constraint graph is the output of $T_0(x)$.

Now, let T_1 be the $(2, \epsilon)$-transformation promised by Lemma 4.4. Let m be the number of constraints in $T_0(x)$, and let $\ell = \log m$. Our final transformation T, then, is $x \mapsto T_1^{(\ell)}(T_0(x))$. Let us verify that T has the required properties. Let $G_0 = T_0(X)$, and for $i = 1, 2, \ldots, \ell$, let $G_i = T_1(G_{i-1})$.

Completeness T_0 takes inputs in L to satisfiable constraint graphs and T_1 take satisfiable constraint graphs to satisfiable constraint graphs. Thus, T takes inputs in L to satisfiable constraint graphs.

Soundness If $x \notin L$, then $\text{UNSAT}(G_0) \geq \frac{1}{m}$. Since T_1 is a $(2, \epsilon)$-transformation, we have by induction that $\text{UNSAT}(G_i) \geq \min\{\frac{2^i}{m}, \epsilon\}$. In particular,

$$\text{UNSAT}(T(x)) = \text{UNSAT}(G_\ell) \geq \min\left\{\frac{2^\ell}{m}, \epsilon\right\}.$$

Since, $2^\ell \geq m$, we conclude that $\text{UNSAT}(T(x)) \geq \epsilon$.

Running time To show that the total time required to compute T is bounded by a polynomial in $|x|$, we first argue that the size of each G_i is bounded by a polynomial in $|x|$. This is clearly true for G_0 because T_0 can be computed in polynomial time. Since T_1 is a linear-size transformation, we have $|G_i| \leq c|G_{i-1}| + d \leq (c+d)|G_{i-1}|$ for some constants c, d. This implies that $|G_i| \leq (c+d)^i |T_0(x)| \leq (c+d)^\ell |T_0(x)| \leq a|x|^b$ for some constants a, b (where the final inequality uses the fact that $\ell = \log |T_0(x)|$.

The total running time is then bounded by the time taken to produce G_0 plus the time taken for the $\ell = O(\log |x|)$ executions of the transform T_1 on inputs of size polynomial in $|x|$. Clearly, the total running time is bounded by a polynomial in $|x|$.

$\blacksquare$

Thus our focus now shifts to Lemma 4.4 and we start to peek into its proof.

4.3 Proof of Lemma 4.4

Dinur proves this lemma by combining two counteracting transformations. The first transformation amplifies the gap by increasing the alphabet size. The second transformation is now in the classical style, which reduces the gap somewhat, but more importantly reduces the alphabet size. While it is clear that both reductions are opposing in direction, the level of detail we have permitted ourselves 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 transformations are especially powerful. The first allows us to amplify the gap by an arbitrary large factor, subject to a sufficiently large explosion in the alphabet size. The second brings the alphabet size to a fixed constant, while paying a fixed price in terms of the gap. These terms are articulated in the assertions below.

Lemma 4.5 (Gap amplification) *For every constant c, there are constants $\epsilon, K > 0$ such that there is a (c, ϵ)-transformation from $\tilde{\mathcal{G}}_{16}$ to $\tilde{\mathcal{G}}_K$.*

In other words, one can pick any factor c to amplify by, and there is a transformation that will achieve that amplification, provided the alphabet size is allowed to increase to an appropriately large constant depending only on c, and independent of the size of the input graph.

Lemma 4.6 (Alphabet reduction) *There exists a constant $\epsilon_2 > 0$, such that for all constants k, there is a $(\epsilon_2, 1)$-transformation from $\tilde{\mathcal{G}}_K$ to $\tilde{\mathcal{G}}_{16}$.*

Here we are buying a small alphabet size by paying for it in terms of a reduction in gap. It is important that this loss in gap, determined by the factor ϵ_2 above, is independent of alphabet size. We will elaborate more on this in a later section.

We devote separate sections to the proof of these two lemmas. For now, we let us show how Lemma 4.4 follows from them.

Proof of Lemma 4.4: Consider the constant ϵ_2 given by Lemma 4.6 and set $c = \frac{2}{\epsilon_2}$. With this value of c, we obtain from Lemma 4.5 constants $\epsilon_1, k > 0$, and a (c, ϵ_1)-transformation T_1 from $\tilde{\mathcal{G}}_{16}$ to $\tilde{\mathcal{G}}_K$. From Lemma 4.6, we obtain a $(\epsilon_2, 1)$-transformation from $\tilde{\mathcal{G}}_K$ to $\tilde{\mathcal{G}}_{16}$. Our final transformation from $\tilde{\mathcal{G}}_{16}$ to $\tilde{\mathcal{G}}_{16}$ is $G \mapsto T_2(T_1(G))$. We claim that T is a $(2, \epsilon_1\epsilon_2)$-transformation. Clearly, T can be computed in polynomial time. Since, for all $G \in \tilde{\mathcal{G}}_{16}$, $|T_1(G)| \leq a_1|G| + b_1$ and for all $G \in \tilde{\mathcal{G}}_K$, $|T_2(G)| \leq a_2|G| + b_2$ (for some constants $a_1, b_1, a_2, b_2 \geq 0$), it follows that and $|T(G)| \leq a_1a_2|G| + a_2b_1 + b_2$; thus, T incurs only a linear blow-up in size. Similarly, since T_1 and T_2 take satisfiable instances to satisfiable instances, so does their composition T. And finally, we have

$$\text{UNSAT}(T(G)) \geq \epsilon_2 \cdot \text{UNSAT}(T_1(G)) \geq \min\{\epsilon_2 \cdot c \cdot \text{UNSAT}(G), \epsilon_2 \cdot \epsilon_1\} = \min\{2 \cdot \text{UNSAT}(G), \epsilon\},$$

where $\epsilon = \epsilon_1\epsilon_2$. This concludes the proof of Lemma 4.4. ▌

With this out of the way, we can turn our attention to *gap amplification* and *alphabet reduction*, and the proofs of Lemmas 4.5 and 4.6

5 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.

5.1 Background: error reduction in randomized computation

The problem of gap amplification in constraint graphs sits in the wider context of error reduction in the developing theory of randomized computation. Since every essentially randomized algorithm errs with some positive probability, it is natural to investigate whether, how, and at what cost this error can be reduced.

Let us examine the problem of gap amplification in constraint graphs in this context. It will be helpful to work in the framework of the PCP arising from constraint graphs as described in Proposition 3.6. What is a Verifier? It is a randomized algorithm, that examines the given proof Π in order to accept or reject. It errs when its computation accepts for some input not in the language. If the gap of the reduction is ϵ then the probability that it errs is at most $1-\epsilon$. So, indeed, to increase the gap, we need to reduce the error in the Verifier's randomized algorithm.

The simplest way to reduce errors in randomized algorithms is by running the algorithm several times independently. For instance, consider one of the classical (randomized) algorithms to determine if an n-bit integer is a prime. The early algorithms (cf. [31]) 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 mistakenly classifying composite numbers as primes) 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 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.)

For gap amplification in PCPs, randomness is a precious resource. After error reduction, if we were to translate the PCP back into a constraint graph, the number of constraints in the resulting graph would be determined by the amount of randomness used by the Verifier of the PCP (as we saw in the proof of Proposition 3.6). Since we want the the size of the constraint graph to increase by at most a constant factor, we can only afford a constant additive increase in the amount of randomness used, so even doubling the amount of randomness is, in general, unaffordable. Is there a more "randomness-efficient" way to reduce errors in randomized algorithms?

This question has been studied extensively in the CS literature under the label of "recycling randomness" [1, 17, 28], and it is known 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 establish the existence of 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., $V(G) = \{0,1\}^n$) 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 extremely well connected, or an "expander" as defined next.

Definition 5.1 (Expander graphs) *For a graph $G = (V, E)$ and and a subset $S \subseteq V$, let $E_S = \{e \in E : \mathsf{tail}(e) \in S \wedge \mathsf{head}(e) \in V \setminus S\}$ denote the set of edges going from S to its complement. The expansion of the set S, denoted $e(S)$, is the quantity*

$$e(S) = \frac{|E_S|}{|S|}.$$

G is said to be an (η, d)-expander if G is d-regular and every set S with $|S| \leq |V|/2$ has expansion $e(S) \geq \eta$. We call an (η, d)-expander G positive *if every vertex of G has at least $\frac{d}{2}$ self-loops.*

It is by now well known in the CS literature that if R' if generated by a k-step random walk on an (η, d)-expander, then the error probability reduces to $2^{-\delta k}$ where δ is a universal constant depending only on η and d. (This result was first shown in a specific context by Ajtai et al. [1], and then noted for its general applicability in [17, 28].) Furthermore, a rich collection of "explicit" (η, d)-expanders have been constructed, allowing for wide ranging application of this insight. We will use the following.

Theorem 5.2 (Existence of expanders) *There exist constants $\eta_0 > 0$ and $d_0 > 0$, such that for all $n \geq 1$, one can in polynomial time in n, construct an (η_0, d_0)-positive expander with n vertices.*

We do not provide a proof of this important theorem. Hoory, Linial and Wigderson [27] survey such constructions, and also describes several applications.

These ideas immediately lead to PCPs where the time complexity and the amount of randomness used by the verifier are as we would like: to amplify some small gap by a factor of 2, we need to run the verifier's algorithm a constant number of times, using a constant number of more random bits than before. There is, however, still one, fatal, flaw in this method. The new Verifier has to make more than two queries to the oracle, whereas, in order to translate the translate the PCP back to a constraint graph (e.g., using Proposition 3.6) we need the verifier to make just *two* queries. Are there ways to amplify the gap, while at the same time maintaining the number of queries made by the Verifier at two?

5.2 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, makes 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 [33] 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 6.2, Part 3.) When combined, these two steps allow us to amplify the gap in PCPs—but now we have lost the "linear size 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 [19] manages to combine the two techniques and achieve the desired gap amplification, and does so with relatively simple proofs. Among other things, Dinur observes that even an example such as the above may not defeat the purpose. For the purposes of Lemma 4.5 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.

5.3 Dinur's gap amplifying transformation

The required transformation T_1 will itself be a composition of three transformations, T_{11}, T_{12} and T_{13}. The first two of these, preprocess the input graph, turning it into a constant degree expander, and preparing the ground for launching T_{13}, which then delivers the required amplification. Both T_{11} and T_{12} lose something in the gap (the alphabet remains the same), but the amplification provided by the T_{13} more than compensate this loss. In Section 5.1, we hinted at the need for conserving randomness while amplifying the gap, and the role of constant-degree expander graphs in reducing the error in randomized algorithms. It turns out that gap amplification in constraint graphs is similarly facilitated if the underlying graph can be assumed to be a constant-degree expander, but the details now are rather more subtle.

5.4 Preprocessing

We are now ready to describe the transformations T_{11} and T_{12}. The first of these, T_{11}, converts an arbitrary constraint graph into a constant-degree graph, losing a fixed factor in the gap.

Lemma 5.3 (Constant-degree constraint graphs) *There are constants $\delta_1, d > 0$, such that for all k, there is a $(\delta_1, 1)$-transformation T_{11} that transforms constraint graphs in $\tilde{\mathcal{G}}_K$ into d-regular constraint graphs in $\tilde{\mathcal{G}}_K$.*

Suppose we are given a $G \in \tilde{\mathcal{G}}_K$ but have no bound on the degrees of the vertices. Consider a vertex v of G whose degree $\deg(v)$ is large. The idea is to split this vertex into $\deg(v)$ vertices, dedicating one vertex for each edge incident on v in the original graph. The difficulty with this naive idea is that the coloring of the new graph could assign different colors to the different copies of v, thereby making the new constraint graph much easier to satisfy. The idea then is to connect the different copies of v using equality constraints so that any coloring that assigns wildly differing colors to the copies of v necessarily violates several of these constraints. We must, however, be careful that the graph stays bounded-degree even with the new equality constraints. A gadget achieving precisely this was invented by Papadimitriou and Yannakakis [32] using expander graphs.

Lemma 5.4 (Equality gadget) *Let H be a constraint graph that is an (η, d)-expander such that every constraint is an equality function. Let $\chi : V(H) \to \Sigma$ be a coloring of H, and let χ^* be the color that appears most frequently in this coloring. Then,*

$$\Pr_{e\in E(H)}[c_e(\chi) = 0] \geq \frac{\eta}{2d} \Pr_{v\in V(H)}[\chi(v) \neq \chi^*].$$

Proof: Consider a color $\sigma \in \Sigma \setminus \{\chi^*\}$. For the set of vertices colored σ, i.e. $\chi^{-1}(\sigma)$, there are at least $\eta|\chi^{-1}(\sigma)|$ edges that have one endpoint in it and one end point outside. The constraint on every such edge evaluates to 0. Since such an edge connects at most two minority color classes, the number of constraints violated by χ is at least $\frac{1}{2}\sum_{\sigma\in\Sigma\setminus\{\chi^*\}} \eta|\chi^{-1}(\sigma)|$. Thus,

$$\begin{aligned} \Pr_{e\in E(H)}[c_e(\chi) = 0] &\geq \frac{1}{2|E|} \sum_{\sigma\in\Sigma\setminus\{\chi^*\}} \eta|\chi^{-1}(\sigma)| \\ &= \frac{1}{2d|V|} \sum_{\sigma\in\Sigma\setminus\{\chi^*\}} \eta|\chi^{-1}(\sigma)| \\ &= \frac{\eta}{2d} \Pr_{v\in V(H)}[\chi(v) \neq \chi^*]. \end{aligned}$$

∎

Using this gadget, we can now prove Lemma 5.3.

Proof of Lemma 5.3: We fill in the details in the outline above. The transformation T_{11} will transform the given graph $G \in \tilde{\mathcal{G}}_K$ into a graph $G' \in \tilde{\mathcal{G}}_K$. Let $K = 2^k$ and $\Sigma(G) = \{0,1\}^k$. The vertex $v \in V(G)$ has $\deg(v)$ different versions in G'. Stated formally, for $v \in V(G)$, we let

$$[v] = \{(v, e) : e \in E(G) \text{ and } \mathsf{tail}(e) = v\}.$$

We will refer to $[v]$ as the *cloud* of v. The vertex set of G' will be the union of all such clouds, that is, $V(G') = \bigcup_{v\in V(G)}[v]$. There are two kinds of edges in G'. First, there are the edges *derived* from the edges of G: for each edge $e \in E(G)$ of the form (u, v), we have an edge $e' \in E(G')$ of the form $((u, e), (v, e^-))$ in G' (recall that e^- is the reversed version of e and $\mathsf{tail}(e^{-1}) = v$); e' is associated with the same constraint as e. Second, we have the edges internal to the clouds, arising out of the Papadimitriou-Yannakakis gadget. Let H_v

be an (η, d_0)-expander graph (as promised by Theorem 5.2) with vertex set $[v]$, where we associate the edge of the form $\{u', v'\}$ with with the equality constraint, now represented by the system of polynomial equations $\{X_{u',i} - X_{v',i} = 0 : 1 \leq i \leq k\}$. Let

$$\begin{aligned} E_1 &= \{e' : e \in E(G)\}; \\ E_2 &= \bigcup_{v \in V(G)} E(H_v). \end{aligned}$$

Thus, $E(G') = E_1 \cup E_2$. Note that all constraints are still expressed as a system of polynomial equations of degree at most two. This describes the transformation T_{11}. It remains to verify that it has the required properties.

It is straightforward to verify that if G is satisfiable, then G' is satisfiable. Let us now argue that $\text{UNSAT}(G') \geq \delta_1 \cdot \text{UNSAT}(G)$, for some absolute constant $\delta_1 > 0$. Fix a coloring χ' of G'. The idea is that if the colors in a single cloud vary substantially, then by Lemma 5.4, a good fraction of the edges inside the cloud will reject. However, if some one color is used heavily in the clouds, then the situation is similar to the original graph, and a good fraction of the constraints between clouds (corresponding to edges derived the original graph) will reject. To state this formally, consider the coloring χ for G obtained from χ' by applying 'majority decoding' on each cloud: $\chi(v)$ is the color that χ' assigns most frequently to the vertices of $[v]$. We call a vertex of G' a *minority* vertex if its color is not shared by the majority (more than half) of the vertices in its cloud. By definition, a fraction $\text{UNSAT}(G)$ of the edges of G are left unsatisfied by the coloring χ'. For an edge $e \in E(G)$, if $c_e(\chi) = 0$, then at least one of the following two conditions must hold: (I) $c_{e'}(\chi') = 0$; (II) e' is incident on a minority vertex of G'. Thus,

$$\begin{aligned} \text{UNSAT}(G) &\leq \Pr_{e \in E(G)}[c_e(\chi) = 0] \\ &\leq \Pr_{e \in E(G)}[c_{e'}(\chi') = 0] + \Pr_{e \in E(G)}[e' \text{ is incident on a minority vertex}]. \end{aligned} \tag{1}$$

Consider the first term on the right. Since $|E(G')| = (d_0 + 1)|E(G)|$, we have

$$\Pr_{e \in E(G)}[c_{e'}(\chi') = 0] \leq (d_0 + 1) \Pr_{e \in E(G')}[c_e(\chi') = 0]. \tag{2}$$

Next consider the second term on the right hand side of (1). This quantity is closely related to the number of minority vertices in G. Indeed, a random vertex v of G' can be generated by first picking a random derived edge and then choosing one of its end points. Thus,

$$\Pr_{v \in V(G')}[v \text{ is a minority vertex}] \geq \frac{1}{2} \Pr_{e \in E(G)}[e' \text{ is incident on a minority vertex}].$$

By Lemma 5.4, it follows that

$$\begin{aligned} \Pr_{e' \in E_2}[c_{e'}(\chi') = 0] &\geq \frac{\eta}{2d_0} \Pr_{v \in V(G')}[v \text{ is a minority vertex}] \\ &\geq \frac{\eta}{4d_0} \Pr_{e \in E(G)}[e' \text{ is incident on a minority vertex}]. \end{aligned}$$

Since, $|E(G')| = (d_0+1)|V| \le 2d_0|V| = 2|E_2|$, we conclude that

$$\Pr_{e\in E(G)}[e' \text{ is incident on a minority vertex}] \le \frac{8d_0}{\eta} \Pr_{e'\in E(G')}[c_{e'}(\chi')=0]. \tag{3}$$

Returning to (1), and using (2) and (3), we obtain

$$\text{UNSAT}(G) \le \left(d_0 + 1 + \frac{8d_0}{\eta}\right) \Pr_{e'\in E(G')}[c_{e'}(\chi')=0].$$

Thus, the claim holds with $\delta_1 = \left(d_0+1+\frac{8d_0}{\eta}\right)^{-1}$ and $d = d_0+1$. ▮

Next, we need a transformation that will turn the constraint graph into an expander.

Lemma 5.5 (Expander constraint graphs) *For all $d > 0$, there are constants $\delta_2, \eta, d_0 > 0$, such that there is a $(\delta_2, 1)$-transformation T_{12} that transforms a d-regular constraint graph in $\tilde{\mathcal{G}}_K$ into a constraint graph in $\tilde{\mathcal{G}}_K$ that is an $(\eta, 2(d+d_0))$-positive expander.*

Proof: Let $G \in \tilde{\mathcal{G}}_K$ be the given d-regular constraint graph. Let H be an (η_0, d_0)-expander (as in Theorem 5.2) with the same set of vertices as G. Let G' be the union of G and H, where the edges of H are associated with the trivial constraint that is always true (formally represented by the empty system of polynomial equations). Now we add $d+d_0$ self-loops (with trivial constraints) to every vertex of G' to obtain the final constraint graph G''. Clearly, if G is satisfiable, then so is G''. Furthermore, since $|E(G)| \ge \left(\frac{d}{2}\right)|V(G)|$ and $|E(G'')| \le 2(d+d_0)|V(G)|$, we have $\text{UNSAT}(G'') \ge \left(\frac{d}{4(d+d_0)}\right)\text{UNSAT}(G)$. So, G'' is an $(\eta_0, 2(d+d_0))$-positive expander, and the map $G \mapsto G''$ is the $(\delta_2, 1)$-transformation, for $\delta_2 = \frac{d}{4(d+d_0)}$. ▮

5.5 The verifier takes a walk

We now approach the crucial part of Dinur's proof.

Lemma 5.6 *For all $c, d, \eta > 0$ there are constants $\epsilon, K > 0$, such that there is a (c, ϵ)-transformation T_{13} taking constraint graphs in $\tilde{\mathcal{G}}_{16}$ that are (η, d)-positive expanders to constraint graphs in $\tilde{\mathcal{G}}_K$.*

Let $G \in \tilde{\mathcal{G}}_{16}$ be an (η, d)-positive expander. Let $\text{UNSAT}(G) = \gamma$, that is, every assignment leaves a fraction γ of the edges unsatisfied. Viewing this in the framework of a PCP (as in Proposition 3.6), the verifier rejects every coloring with probability at least γ. The natural way to increase this rejection probability is for the verifier to examine several constraints at the same time. There are two concerns: (1) picking each constraint requires us to toss new coins, and if these are done independently, the size the constraint graph we derive from the resulting PCP (by invoking Proposition 3.6) will not be linear in the size of original graph; (2) this strategy requires the verifier to examine the colors assigned to several vertices, but in order to obtain a constraint graph in the end, we can allow at most two probes by the verifier. The first concern is addressed by picking the edges, not independently, but by taking a random walk on an expander graph. The length of the random walk will depend on the number of edges we want to examine, and the amplification we wish to achieve. Since we only need amplification by a constant factor, walks of constant length will suffice. Now, let us address the second concern. Given that the edges we pick are going to be in the vicinity of the starting and ending vertices, we expand the alphabet of vertices, so that at each vertex we now have information about the colors assigned to all vertices in its neighborhood, that is, every vertex u in the neighborhood v has an *opinion* about v's color. There is no guarantee, however, that these opinions are consistent. How, then, can we conclude that examining several edges in this manner is somehow related to examining the these edges in the original graph? For this, we must look deeper into Dinur's proof.

Based on G, we will first describe a verifier who reads two locations from a proof, and simultaneously checks the constraints for several edges of the original graph. This will amplify the gap. We will then transform this PCP into a new constraint graph with a larger alphabet, as we did in the proof of Proposition 3.6

5.6 The PCP

Fix $t \geq 1$. We describe the proof oracle and verifier below. The verifier probes this proof randomly at two locations, and based on the values read, decides to accept or reject. If the original constraint graph is satisfiable, then there exists a proof that the verifier accepts with probability 1. On the other hand, if the original constraint graph is ϵ-far from satisfiable, then the verifier rejects every proof with probability at least $\epsilon \cdot \Omega(t)$.

The proof: For each vertex $v \in V(G)$, the proof now provides an assignment for all vertices that are within a distance t from v. We view the proof as a function $\mathcal{A} : V(G) \to \Sigma(G)^{(d+1)^t}$, where $\mathcal{A}(v)$ denotes this (partial) assignment provided at vertex v. We use $\mathcal{A}(v)[w]$ to refer to the value $\mathcal{A}(v)$ assigns to vertex w, and think of $\mathcal{A}(v)[w]$ as vertex v's *opinion* for the color that should be assigned to w in order to satisfy G. Thus, every vertex within distance t of w has an opinion for w; there is no guarantee, however, that these opinions agree with each other. Vertices w that don't appear within distance t of v are not explicitly assigned a value in $\mathcal{A}(v)$; for such a vertices w, we say that $\mathcal{A}(v)[w]$ is null. Let A_1 and A_2 be two partial assignments, and let e be an edge of G of the form (u, v). We say that A_1 and A_2 pass the test at e, if at least one of the following conditions holds: (i) one of $A_1(u)$, $A_1(v)$, $A_2(u)$, and $A_2(v)$ is null; (ii) A_1 and A_2 agree on $\{u, v\}$ and $c_e(A_1[u], A_2[v]) = 1$.

The verifier: The verifier picks two random vertices, $\mathbf{a}$ and $\mathbf{b}$, of the graph and performs a test on the values stored there.

The random walk: The two vertices $\mathbf{a}$ and $\mathbf{b}$ are generated using a random walk, as follows.

I. Let $\mathbf{a} = \mathbf{v}_0$ be a random vertex chosen uniformly from $V(G)$. Repeat Step II until some condition for stopping is met.

II. Having chosen $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{i-1}$, let $\mathbf{e}_i$ be a random edge leaving $\mathbf{v}_{i-1}$, chosen uniformly among the d edges e with $\mathsf{tail}(e) = \mathbf{v}_{i-1}$. Let $\mathbf{v}_i$ be the other end point of $\mathbf{e}_i$, that is, $\mathbf{v}_i = \mathsf{head}(\mathbf{e}_{i-1})$. With probability $\frac{1}{t}$, STOP and set $\mathbf{T} = i$.

The test: Suppose the random walk visits the vertices $\mathbf{a} = \mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_T = \mathbf{b}$ using the sequence of edges, $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_T$. If $\mathcal{A}(\mathbf{a})$ and $\mathcal{A}(\mathbf{b})$ fail (i.e. don't pass) the test at some $\mathbf{e}_i$, the verifier rejects; otherwise, she accepts. When $\mathbf{a}$ and $\mathbf{b}$ are clear from the context, we say that *the test at* $\mathbf{e}_i$ *fails,* when we mean that $\mathcal{A}(\mathbf{a})$ and $\mathcal{A}(\mathbf{b})$ fail the test at $\mathbf{e}_i$.

Lemma 5.7 *Suppose the constraint graph G is an (η, d)-expander.*

(a) If G is satisfiable, then there is a proof that the verifier accepts with probability 1.

(b) If G is ϵ-far from satisfiable, then the verifier rejects every proof with probability at least

$$\left(\frac{1}{512C}\right) \cdot t \cdot \min\left\{\epsilon, \frac{1}{t}\right\},$$

where $C = 1 + \frac{d^2}{\eta^2}$.

Proof: Part (a) is straightforward. Given a satisfying assignment A for G, let the proof be the assignment $\mathcal{A}$ such that $\mathcal{A}(v)[w] = A(w)$.

The idea for part (b) is the following. Fix an assignment $\mathcal{A}$. We will argue that for such an assignment $\mathcal{A}$ to succeed in convincing the verifier, the opinions of different vertices must be generally quite consistent. This suggests that a good fraction of $\mathcal{A}$ is consistent with a fixed underlying assignment A for G. Now, since G is ϵ-far from satisfiable, A must violate at least an fraction ϵ of the constraints in G. Since the verifier examines t edges on an average, the expected number of unsatisfied edges she encounters is $t\epsilon$. Most of the work will go into showing that when she does encounter these edges, she rejects with a sufficient probability and that these rejections are not concentrated on just a few of her walks. In our analysis we will use the following fact, which formalizes the memoryless nature of the verifier's random walk. (We omit the proof.)

Lemma 5.8 (Fact about the random walk) *Let $e \in E(G)$ be of the form (u, v). Consider the verifier's walks conditioned on the event that the edge e appears exactly k times (for some $k \geq 1$) in the walk, that is, the number of i's for which $\mathbf{e}_i = e$ (in particular, $v_{i-1} = u$ and $v_i = v$) is exactly k. Conditioned on this event, consider the starting vertex, $\mathbf{a}$, and the ending vertex, $\mathbf{b}$. We claim that $\mathbf{a}$ and $\mathbf{b}$ are independent random variables. Furthermore, $\mathbf{a}$ has the same distribution as the random vertex obtained by the following random process.*

> *Start the random walk at u, but stop with probability $\frac{1}{t}$* before *making each move (so we stop at u itself with probability $\frac{1}{t}$). Output the final vertex.*

Similarly, we claim that $\mathbf{b}$ *can be generated using a random walk starting from v and stopping with probability $\frac{1}{t}$ before each step.*

Now, fix a proof $\mathcal{A}$. Let us "decode" $\mathcal{A}$ and try to obtain an assignment A for G. The idea is to define $A(u)$ to be most popular opinion available in $\mathcal{A}$ for u, but motivated by Lemma 5.8, the popularity of an opinion will be determined by considering a random walk.

The new assignment A for G: To obtain $A(u)$, we perform a random walk starting from u mentioned in Lemma 5.8 (stopping with probability $\frac{1}{t}$ before each step). Restrict attention to those walks that stop within $t-1$ steps. Let the vertex where the walk stops be $\mathbf{b}_u$. This generates a distribution on the vertices of G. For each letter σ in the alphabet, determine the probability (under this distribution) that $\mathbf{b}_u$'s opinion for u is σ. Then, let $A(u)$ be the letter that has the highest probability. Formally,

$$A(u) \stackrel{\Delta}{=} \arg\max_{\sigma\in\Sigma} \Pr[\mathcal{A}(\mathbf{b}_u)[u] = \sigma \text{ and } T \leq t-1].$$

We now relate the verifier's probability of rejection to the the fraction of edges of G left unsatisfied by A. Since G is ϵ-far from satisfiable, an fraction ϵ of the edges of G are left unsatisfied by A. We wish to argue that whenever the verifier encounters one of these edges in her walk, she is likely to reject the walk. Let F be a subset of these unsatisfied edges of the largest size such that $|F| \leq \frac{1}{t}$. Then,

$$\min\left\{\epsilon, \frac{1}{2t}\right\} \leq \frac{|F|}{|E|} \leq \frac{1}{t}. \tag{4}$$

Now, consider the edges used by the verifier in her walk: $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_\mathbf{T}$.

Definition 5.9 (Faulty edge) *We say that the i-th edge of the verifier's walk is faulty if*

- $\mathbf{e}_i \in F$ *and*
- $\mathcal{A}(\mathbf{a})$ *and* $\mathcal{A}(\mathbf{b})$ *fail the test at* $\mathbf{e}_i$.

Let $\mathbf{N}$ *be the random variable denoting the number of faulty edges on the verifier's walk.*

Since the verifier rejects whenever she encounters a faulty edge on her walk, it is enough to show that $\mathbf{N} > 0$ with high enough probability. We will prove the following two claims below.

Claim 5.10

$$\mathbf{E}[\mathbf{N}] \;\geq\; t\frac{|F|}{8|E|} \tag{5}$$

$$\textit{and} \qquad \mathbf{E}[\mathbf{N}^2] \;\leq\; Dt\frac{|F|}{|E|}, \tag{6}$$

where $D = 4\left(1 + \frac{d^2}{\eta^2}\right)$.

Let us now assume that these claims hold, and complete the proof of the Lemma 5.7:

$$\begin{aligned}
\Pr[\textit{verifier rejects}] \geq \Pr[\mathbf{N} > 0] \;&\geq\; \frac{\mathbf{E}[\mathbf{N}]^2}{\mathbf{E}[\mathbf{N}^2]} \\
&\geq\; \left(\frac{1}{64D}\right) \cdot t \cdot \left(\frac{|F|}{|E|}\right) \\
&\geq\; \left(\frac{1}{128D}\right) \cdot t \cdot \min\left\{\epsilon, \frac{1}{t}\right\}.
\end{aligned}$$

For the second inequality, we used the fact (which follows from the Chebyshev-Cantelli inequaltiy, see also, Alon and Spencer [4, Section 4.8, Ex. 1]) that for any non-negative random variable $\mathbf{X}$, $\Pr[\mathbf{X} > 0] \geq \frac{\mathbf{E}[\mathbf{X}]^2}{\mathbf{E}[\mathbf{X}^2]}$. For the last inequality we used (4). ▌

5.7 Proofs of the claims

Proof of (5): We will estimate the expected number of faulty occurrences for each edge in F. Fix one such edge e of the form (u, v), and let $\mathbf{N}_e$ denote the number of *faulty* occurrences of e in the verifier's walk. Let $\#e$ denote the number of occurrences (not necessarily faulty) of e in the walk. Note that the i-th edge of the walk is uniformly distributed over the set of all edges, and so $\mathbf{E}[\#e] = \frac{t}{|E|}$ for all e. Condition on the event $\#e = k$, and consider the starting vertex $\mathbf{a}$ and the ending vertex $\mathbf{b}$. By Lemma 5.8, $\mathbf{a}$ and $\mathbf{b}$ can be generated using independent lazy random walks starting at u and v respectively. The probability that the walk to generate $\mathbf{a}$ traverses t or more edges is $\left(1 - \frac{1}{t}\right)^t \leq \exp(-1)$. Thus, with probability at least $\alpha \overset{\Delta}{=} 1 - \exp(-1)$ the starting vertex $\mathbf{a}$ is at a distance at most $t - 1$ from u, and hence at most t from v. Let p_u be the probability that the $\mathbf{a}$ is at a distance at most $t - 1$ from u *and* $\mathcal{A}(\mathbf{a})[u] = A(u)$; similarly, let p_v be the probability that $\mathbf{b}$ is at a distance at most $t - 1$ from v and $\mathcal{A}(\mathbf{b})[v] = A(v)$. Now, the test at e fails if $\mathcal{A}(\mathbf{a})[u] \neq \mathcal{A}(\mathbf{b})[u]$ (and are both not null). This happens with probability at least $\alpha(\alpha - p_u)$. Similarly, by considering v, we conclude that the test at e fails with probability at least $\alpha(\alpha - p_v)$. Furthermore, with probability at least $p_u p_v$, we have

$$c_e(\mathcal{A}(\mathbf{a})[u], \mathcal{A}(\mathbf{b})[v]) = c_e(A(u), A(v)) = 0,$$

in which case the test at e fails. Thus, overall,

$$\Pr[\text{the test at } e \text{ fails} \mid \#e = k] \geq \max\{\alpha(\alpha - p_u), \alpha(\alpha - p_v), p_u p_v\} \geq \alpha^2\left(\frac{\sqrt{5} - 1}{2}\right)^2 > \frac{1}{8}.$$

If $\mathcal{A}(\mathbf{a})$ and $\mathcal{A}(\mathbf{b})$ fail the test at e, then all the k occurrences of e in the walk are considered faulty. Thus,

$$\begin{aligned}
\mathbf{E}[\mathbf{N}_e] &= \sum_{k>0} \Pr[\mathbf{N}_e = k] \cdot k \\
&\geq \sum_{k>0} \Pr[\#e = k] \cdot k \cdot \Pr[\text{the test at } e \text{ fails} \mid \#e = k] \\
&\geq \sum_{k>0} \Pr[\#e = k] \cdot k \cdot \left(\frac{1}{8}\right) \\
&= \left(\frac{1}{8}\right) \mathbf{E}[\#e] \\
&= \frac{t}{8|E|}.
\end{aligned}$$

Finally, summing over all $e \in F$, we have

$$\mathbf{E}[\mathbf{N}] = \sum_{e} \mathbf{E}[\mathbf{N}_e] \geq \left(\frac{|F|}{8|E|}\right) t.$$

$\blacksquare$

We have shown that the expected number of faulty edges on the verifier's walk is large. However, this does not automatically imply that the number of faulty edges is positive with reasonable probability, for it could be that faulty edges appear on the verifier's walk in bursts, and just a few walks account for the large expectation. This is where we use the fact that our underlying graph is an expander. Intuitively, one expects that a random walk in an expander graph is not likely to visit the small set of edges, F, too many times. The following proposition quantifies this intuition by showing that in the random walk the events of the form "$\mathbf{e}_i \in F$" are approximately pairwise independent.

Proposition 5.11 (see Dinur [19]) *For $j > i$,*

$$\Pr[\mathbf{e}_j \in F \mid \mathbf{e}_i \in F] \leq \left(1 - \frac{1}{t}\right)^{j-i} \left(\frac{|F|}{|E|} + \left(1 - \frac{\eta^2}{d^2}\right)^{j-i-1}\right).$$

A proof of this proposition appears in the Appendix B.

Proof of (6): Let χ_i be the indicator random variable for the event "$\mathbf{e}_i \in F$". Then, $\Pr[\chi_i = 1] = \frac{|F|}{|E|}\left(1 - \frac{1}{t}\right)^{i-1}$, and

$$\mathbf{E}[\mathbf{N}^2] \leq \left(\sum_{i=1}^{\infty} \chi_i\right)^2$$

$$\begin{aligned}
&\leq 2 \sum_{1\leq i\leq j<\infty} \mathbf{E}[\chi_i\chi_j] \\
&\leq 2\sum_{i=1}^{\infty} \Pr[\chi_i = 1] \sum_{j\geq i} \Pr[\chi_j = 1 \mid \chi_i = 1] \\
&\leq 2\sum_{i=1}^{\infty} \Pr[\chi_i = 1] \left[1 + \sum_{\ell\geq 1} \left(1-\frac{1}{t}\right)^{\ell} \left(\frac{|F|}{|E|} + \left(1-\frac{\eta^2}{2d^2}\right)^{\ell-1}\right)\right] \qquad (7) \\
&\leq 2\sum_{i=1}^{\infty} \Pr[\chi_i = 1] \left[1 + \sum_{\ell\geq 1} \left(1-\frac{1}{t}\right)^{\ell} \frac{|F|}{|E|} + \left(1-\frac{\eta^2}{2d^2}\right)^{\ell-1}\right] \\
&\leq 2t\frac{|F|}{|E|}\left(1 + t\frac{|F|}{|E|} + \frac{2d^2}{\eta^2}\right),
\end{aligned}$$

where we used Proposition 5.11 in (7). Claim (6) follows from this because we have assumed (see (4) above) that $\frac{|F|}{|E|} \leq \frac{1}{t}$. ▌

5.8 The final constraint graph

It is relatively straightforward to model the PCP described above as a constraint graph. There is one technicality that we need to take care of: the verifier's walks are not bounded in length, and a naive translation would lead to a graph with infinitely many edges. We now observe that we can truncate the verifier's walk without losing much in the rejection probability.

Verifier with truncated walks: We will show that a version of Lemma 5.7 holds even when the verifier's walks are truncated at $T^* = 5t$, and she just accepts if her walk has not stopped within these many steps.

Lemma 5.12 (Truncated verifier) *Suppose the constraint graph G with alphabet Σ is an (η, d)-expander. Consider the verifier with truncated walks.*

(a) If G is satisfiable, then there is a proof that the verifier accepts with probability 1.

(b) If G is ϵ-far from satisfiable, then the verifier rejects every proof with probability at least

$$\left(\frac{1}{1024C}\right) \cdot t \cdot \min\left\{\epsilon, \frac{1}{t}\right\},$$

where $C = 2\left(1 + (\frac{d}{\eta})^2\right)$.

Proof: We only show how the previous proof is to be modified in order to justify this lemma. If the verifier's walk is truncated before stopping, then no edge on the walk is declared faulty. Under this definition, let $\mathbf{N}'$ be the number of faulty edges in the verifier's random walk ($\mathbf{N}' = 0$ whenever the walk is truncated). Let us redo Claim (5). Let $\mathcal{I}\{\mathbf{T} \geq T^* + 1\}$ be the indicator random variable for the event $T \geq T^* + 1$. Then, $\mathbf{N}' = \mathbf{N} - \mathbf{N} \cdot \mathcal{I}\{\mathbf{T} \geq T^* + 1\}$, and

$$\mathbf{E}[\mathbf{N}'] = \mathbf{E}[\mathbf{N}] - \mathbf{E}[\mathbf{N} \cdot \mathcal{I}\{\mathbf{T} \geq T^* + 1\}]. \tag{8}$$

We already have a lower bound for $\mathbf{E}[\mathbf{N}]$ in Claim (5), so it is sufficient to obtain an upper bound for the second term on the right, which accounts for the contribution to $\mathbf{E}[\mathbf{N}]$ from long walks.

The contribution to $\mathbf{E}[\mathbf{N}]$ from walks of length ℓ is at most $\ell|F|/|E|$ times the probability that $\mathbf{T} = \ell$. So, the contribution to $\mathbf{E}[\mathbf{N}]$ from walks of length at least $T^* + 1$ is

$$\Pr[\mathbf{T} \geq T^* + 1] \cdot \mathbf{E}[\mathbf{T} \mid \mathbf{T} \geq T^* + 1] \cdot \frac{|F|}{|E|}.$$

The first factor is at most $(1 - \frac{1}{t})^{T^*}$, the second is $T^* + t$. So,

$$\mathbf{E}[\mathbf{N} \cdot \mathcal{I}\{\mathbf{T} \geq T^* + 1\}] \leq \exp\left(-\frac{T^*}{t}\right)(T^* + t) \cdot \frac{|F|}{|E|} \leq t \cdot \frac{|F|}{|E|} \cdot \frac{1}{16}.$$

Thus, from (8), we obtain

$$\mathbf{E}[\mathbf{N}'] \;\geq\; t \cdot \frac{|F|}{|E|} \cdot \frac{1}{8} - t \cdot \frac{|F|}{|E|} \cdot \frac{1}{16} \;\geq\; t \cdot \frac{|F|}{|E|} \cdot \frac{1}{16}.$$

Note that $\mathbf{N}' \leq \mathbf{N}$, so the upper bound in (6) applies to $\mathbf{E}[\mathbf{N}']$ as well. Now, Lemma 5.12 follows from the inequality $\mathbf{E}[\mathbf{N}' > 0] \geq \mathbf{E}[\mathbf{N}']^2/\mathbf{E}[\mathbf{N}'^2]$. ▌

Definition 5.13 (The product constraint graph) *Let G be a d-regular constraint graph with alphabet Σ. The product graph G^t is defined as follows.*

- *The vertex set of the graph G^t is $V(G)$.*
- *The alphabet for G^t is $\Sigma^{(d+1)^t}$.*
- *The edges and their constraints correspond to the verifier's actions outlined above. We imagine that the verifier's moves are described by a random string of length T^* over the set $[d] \times [t]$. The first component determines which outgoing edge the verifier takes, and she stops after that step if the the second component is* 1 *(say). For each vertex a and each sequence τ, we have a an edge labeled τ leaving (with tail) a, corresponding to the walk starting from vertex a determined by τ.*
- *If the walk does not terminate at the end of τ, then the ending vertex (i.e. head) of this edge is also a (it is a self-loop), and the constraint on that edge is always satisfied.*

- *If the walk does terminate, and the final vertex is b, then the edge labeled τ has the form (a,b), and its constraint is the conjunction of all constraints of G checked by the verifier along this walk. Note that in G the constraints were represented by a system of polynomial equations of degree at most two, then the constraints in G^t also have the same form.*

Thus, every vertex has $(dt)^{T^*}$ edges leaving it, and the total number of edges in G^t is exactly $|V(G)| \cdot (dt)^{T^*}$.

Lemma 5.14 *For all constants t, the function $T_{13} : G \mapsto G^t$ (where G is a restricted constraint graph that is an (η, d)-positive expander) is a $\left(\frac{t}{C_2'}, \frac{1}{t}\right)$-transformation, where $C_2' = O\left((\frac{d}{\eta})^2\right)$. Also, $|\Sigma(G^t)| = |\Sigma(G)|^{(d+1)^t}$; in particular, $\Sigma(G^t)$ depends only on the alphabet of the original graph and the parameter t. The constraints of G^t are represented as a system of polynomial equations of degree at most two; thus, $G^t \in \tilde{\mathcal{G}}_K$ where $K = |\Sigma(G^t)|$.*

Lemma 5.6 follows immediately from this.

6 Alphabet reduction

Recall that in this section we wish prove to Lemma 4.6, reproduced below.

Lemma 4.6 *There exists a constant $\epsilon_2 > 0$, such that for all constants k, there is a $(\epsilon_2, 1)$-transformation from $\tilde{\mathcal{G}}_K$ to $\tilde{\mathcal{G}}_{16}$.*

For the purposes of this section it will be useful to consider a further generalization of the constraint graph coloring problems to "hypergraph constraints". This definition is motivated by the fact that the proof of Lemma 4.6 will reduce graph coloring on arbitrary alphabets to hypergraph coloring over a fixed alphabet and then convert this back to a constraint graph coloring problem.

The constraint hypergraph coloring problem is a generalization of the constraint graph coloring problem to r-uniform hypergraphs with each constraint applying to edges to the hypergraph. The following definition captures this notion formally.

Definition 6.1 (Constraint hypergraph, coloring) *For a positive integer r, an r-*constraint hypergraph *G is a tuple $\langle V, E, \Sigma, \mathcal{C}\rangle$, where (V, E) is an r-uniform hypergraph (we allow multiple edges and multiple incidences of the same vertex in an edge), Σ is a finite set called the alphabet of G, and $\mathcal{C}$ is a collection of constraints, $\langle c_e : e \in E\rangle$, where each c_e is a function from $\Sigma^r \to \{0,1\}$. A* coloring *of G is a function $A : V \to \Sigma$. We say that the coloring A satisfies an edge e of the form $(u_1, \ldots, u_r)$ if $c_e(A(u_1), \ldots, A(u_r)) = 1$. We say that the coloring A satisfies G, if A satisfies all edges in G. If there is a coloring that satisfies G,* *then we say that G is satisfiable. We say that G is ϵ-far from satisfiable if every coloring leaves at least a fraction ϵ of the edges of G unsatisfied. Let*

$$\text{UNSAT}(G) \;=\; \max\{\epsilon : G \textit{ is } \epsilon\textit{-unsatisfiable}\} \;=\; \min_A \frac{|\{e : A \textit{ does not satisfy } e\}|}{|E|}.$$

We use $\mathcal{G}_K^{(r)}$ to denote the set of r-constraint graphs with an alphabet of size K. When $K = 2^k$, $\Sigma = \mathbb{F}_2^k$ and the constraints c_e are given by a system of quadratic (degree two) polynomials over the rk variables from $\mathbb{F}_2$ representing the colors of the r underlying variables, we say that the constraint hypergraph is a restricted constraint hypergraph. We use $\tilde{\mathcal{G}}_K^{(r)}$ to denote the set of restricted constraint graphs

To motivate our main lemmas we start by describing some of the basic (and well-known) reductions between constraint hypergraph coloring problems. These results don't really suffice to prove Lemma 4.6 but thinking about this proposition provides a good warmup to thinking about reductions. It also emphasizes the novelty in the lemma.

Proposition 6.2 *Fix integers $K = 2^k$ and $r > 0$. Then the following are true.*

1. *There exists a $(1, 1)$-transformation from $\tilde{\mathcal{G}}_K^{(r)}$ to $\tilde{\mathcal{G}}_2^{(rk)}$.*
2. *There exists a $(1/2^{r+2}, 1)$-transformation from $\tilde{\mathcal{G}}_2^{(r)}$ to $\tilde{\mathcal{G}}_2^{(3)}$.*
3. *There exists a $(1/r, 1)$-transformation from $\tilde{\mathcal{G}}_K^{(r)}$ to $\tilde{\mathcal{G}}_{K^r}$.*

Proof sketch: We won't really prove Parts 1 and 2, since they are somewhat tangential to our goal, and furthermore quite elementary. But we will provide some hints, before giving a somewhat more detailed sketch of the proof of Part 3. The first part is easy to achieve by simply viewing every vertex of an instance $\tilde{\mathcal{G}}_K^{(r)}$ as k vertices from an instance of $\tilde{\mathcal{G}}_2^{(rk)}$. A 2^k-coloring can then be viewed as binary coloring of the k corresponding vertices and constraints now apply to $r \cdot k$ vertices at a time. The properties of the transformation can now be verified easily. The second part follows from the classical reduction from r-SAT to 3-SAT (due to [18]). Note that this reduction can be set up to create only restricted instances.

Finally, the third part: This part is based on a result of Fortnow, Rompel and Sipser [23]. Consider an instance G of $\tilde{\mathcal{G}}_K^{(r)}$ with vertices V, edges E, on alphabet Σ and constraints $\langle c_e \rangle_{e \in E}$. The vertices of the transformed instance G' will consist of a vertex u' for every vertex $u \in V$ and e' for every edge $e \in E$. Colorings A' to the vertices of G' take values in the set Σ^r. Let $A'(u')_i$ denote the ith coordinate of such a coloring (which lies in Σ). Corresponding to a constraint $c_e(u_1, \ldots, u_r)$ of G, the instance G' will consist of r constraints $c'_{e,i}$, one for each $i \in [r]$. The constraint $c'_{e,i}$ applies to the variable u'_i and e' and is satisfied by an assignment A' if and only if $c_e(A'(e')) = 1$ and $A'(e')_i = A'(u'_i)_1$. It is easy to see that the transformation preserves linear size (the number of constraints of G' is r times the number of constraints of G). It can also be verified that if A is an assignment to the vertices of G satisfying every constraint of G, then any assignment satisfying $A'(u')_1 = A(u)$ and $A'(e) = (A(u_1), \ldots, A(u_r))$ for $e = \langle u_1, \ldots, u_r \rangle$ satisfies every constraint of G'. Finally given an assignment A' to the instance G' consider the assignment $A(u) = A'(u')_1$ for every $u \in G$. We claim that if A' fails to satisfy a fraction ϵ/r of the constraints of G' then A fails to satisfy only a fraction ϵ of the constraints of G. To see this it suffices to note that if for some $e \in E$ the constraints $c'_{e,i}$ are satisfied by A' for every $i \in [r]$ then c_e is satisfied by A since

$$\begin{aligned} c_e(A(u_1), \ldots, A(u_r)) &= c_e(A'(u'_1)_1, \ldots, A'(u'_r)_1) \\ &= c_e(A'(e')_1, \ldots, A'(e')_r) \\ &= 1, \end{aligned}$$

where the first equality is by the definition of A, the second by the second condition in the satisfiability of (all of) the $c'_{e,i}$'s and the third equality from the first condition of the satisfiability of (any of) the $c'_{e,i}$'s. ▌

The above proposition gives a rich collection of simple transformations between constraint hypergraph coloring problems. Indeed if we put together Parts 1, 2, and 3 of Proposition 6.2 above, we get a transformation from $\tilde{\mathcal{G}}_K$ to $\tilde{\mathcal{G}}_{2^3}$, which is one of the goals of Lemma 4.6. The weakness of Proposition 6.2 becomes apparent when we consider the guarantees we get from it. A straightforward application yields a $(1/(3 \cdot 2^{2k+2}), 1)$-transformation from $\tilde{\mathcal{G}}_K$ to $\tilde{\mathcal{G}}_8$ which fails the promise that the gap of the reduction would reduce by a factor independent of k. It is this aspect that makes Lemma 4.6 quite non-trivial, and forces a significant departure from standard transformations of this nature. Fortunately work done in the 1990's (in particular, the work of Arora et al. [6]) gives a much stronger version which suffices for our purposes. We paraphrase their result in our language here.

Lemma 6.3 *There exists an $\epsilon_3 > 0$ such that for all integers K there exists an $(\epsilon_3, 1)$-transformation from $\tilde{\mathcal{G}}_K$ to $\tilde{\mathcal{G}}_2^{(4)}$.*

Proving this lemma will occupy us for the rest of this section. But first we note that this lemma immediately yields Lemma 4.6.

Proof of Lemma 4.6: We compose the transformation from Lemma 6.3 with the one from Part 3 of Proposition 6.2. The composition yields a $(\epsilon_3/4, 1)$-transformation, where ϵ_3 is the constant of Lemma 6.3, thus yielding Lemma 4.6 for $\epsilon_2 = \epsilon_3/4$. ▌

We now focus on the task of transforming $\tilde{\mathcal{G}}_K$ to $\tilde{\mathcal{G}}_2^{(4)}$. The steps in the transformation will be similar to the steps of the transformation given above in the proof of Part 1 of Proposition 6.2 (only each step will now be significantly more complex). We will start by "transforming" the vertices, explaining how to interpret colorings of one instance as colorings of the other. Then we will transform the constraints of the source instance. Analyzing this process will then yield the reduction. The following two sections (Sections 6.1 and 6.2) introduce some of the machinery. Section 6.3 presents the actual transformation which is then analyzed in Section 6.4.

6.1 Encodings of vertex colorings

In the proof of Part 1 of Proposition 6.2, we transformed $K = 2^k$ coloring to 2 coloring (of hypergraphs). This transformation converted a single vertex u of $G \in \tilde{\mathcal{G}}_K^{(r)}$ into k vertices $u_1, \ldots, u_k$ of $G' \in \tilde{\mathcal{G}}_2^{(rk)}$. The colorings $A(u)$ and $A'(u_1), \ldots, A'(u_k)$ were thus in one to one correspondence with each other allowing us to compare colorings of G' with colorings of G. While this made the analysis simple, it also introduced essential weaknesses in the transformation. In Part 1 of Proposition 6.2, this led to making the edges of the hypergraph larger by a factor of k. If we didn't want to pay such a price and keep the hyperedges small, then the reduction would have lost at least $1/k$ in its strength. While we won't go into details, this loss boils down to the fact that two different colorings $A_1(u) = a$ and $A_2(u) = a'$ might only be discernible only by considering the colors of one of the k corresponding vertices $u_1, \ldots, u_k$. In turn this weakness boils down to the fact that in translating the K-coloring of G to a 2-coloring of G' we adopted a simplistic representation of elements of K as binary strings—one which was not very good at distinguishing distinct elements of K. As we will see shortly, one can do much better by picking better, so-called "error-correcting" representations.

Below we will refer to a function $E : \{0,1\}^k \to \{0,1\}^\ell$ as an "encoding" of elements of $\{0,1\}^k$. Corresponding to an encoding function, we will also consider decoding functions $D : \{0,1\}^\ell \to \{0,1\}^k$. Relative to such encoding functions our transformations from $G \in \tilde{\mathcal{G}}_K$ to $G' \in \tilde{\mathcal{G}}_2^{(4)}$ will proceed by creating ℓ vertices $u_1, \ldots, u_\ell$ in G' for each vertex $u \in G$. (There will be additional vertices, and of course some edges and constraints, corresponding to the constraints of G, but we will describe those later.) The intent will be to show that if G is satisfied by the coloring A, then G' is satisfied by a coloring A' which assigns to the vertex u_i the color $E(A(u))_i$. Similarly when it comes to analyzing unsatisfiable instances we will infer properties of a coloring A' to the vertices of G' by consider the induced coloring A given by $A(u) = D(A'(u_1), \ldots, A'(u_\ell))$. Motivated by this intention we now describe the encoding and decoding function we will use, and some interesting properties of this encoding function.

Definition 6.4 (The Hadamard encoding) *The Hadamard Encoding H maps strings of length k to strings of length 2^k. The encoding $H(a)$ of a string $a \in \{0,1\}^k$ is itself indexed by strings $w \in \{0,1\}^k$ with the wth coordinate of $H(a)$, denoted $H(a)(w)$, being given by* $\sum_{i=1}^k a_i w_i \bmod 2$.

For the decoding we will use a canonical decoding function. For strings $x, y \in \{0,1\}^\ell$ let $\delta(x,y) = \frac{|\{i | x_i \neq y_i\}|}{\ell}$. We say x is δ-close to y if $\delta(x,y) \leq \delta$.

The Hadamard decoding of a string $x \in \{0,1\}^{2^k}$, denoted $H^{-1}(x) = a$ where a minimizes $\delta(H(a), x)$ with ties being broken arbitrarily. In what follows we will show some nice properties of the Hadamard encoding (and decoding) function.

The properties of the Hadamard encoding in turn go back to the properties of linear functions. From this point onward we will view the elements $\{0,1\}$ as elements of the field $\mathbb{F}_2$ and $\{0,1\}^k$ as a k dimensional vector space over $\mathbb{F}_2$. In particular for $x, y \in \{0,1\}^k$ we will let $x + y$ denote their vector sum in $\mathbb{F}_2^k$.

Definition 6.5 (Linear functions) *A function $f : \{0,1\}^k \to \{0,1\}$ is called a linear function if it satisfies $f(x+y) = f(x)+f(y)$ for every $x, y \in \{0,1\}^k$ (and where $x+y$ denotes the vector addition of two vectors over $\mathbb{F}_2$. Equivalently f is linear if it is a homogeneous polynomial of degree 1 in its arguments, i.e., there exists $c_1, \ldots, c_k$ so that $f(x_1, \ldots, x_k) = \sum c_i x_i$.*

The properties of linear functions (and other low-degree functions) are derived from the following simple and well-known results about low-degree multivariate polynomials.

Lemma 6.6 ([34, 36]) *Suppose $p(Z) \in \mathbb{F}[Z_1, Z_2, \ldots, Z_n]$ is a non-zero polynomial of degree at most d over a field $\mathbb{F}$. Let $S \subseteq \mathbb{F}$ be a finite set. Then, $\Pr_{z\in S^n}[p(z) = 0] \leq \frac{d}{|S|}$.*

Proof: We use induction on n. For $n = 1$, the claim is obvious because a non-zero polynomial can have at most d roots. Assume Z_n appears in the polynomial (otherwise, $p(Z) \in \mathbb{F}_2[Z_1, Z_2, \ldots, Z_{n-1}]$ and the claim follows from the induction hypothesis) and the degree in Z_n is d_1. Write

$$p(Z) = Z_n^{d_1} \cdot p_1(Z_1, Z_2, \ldots, Z_{n-1}) + p_2(Z_1, Z_2, \ldots, Z_n),$$

where $p_1 \in \mathbb{F}_2[Z_1, Z_2, \ldots, Z_{n-1}]$ is a non-zero polynomial of degree at most $d - d_1$. By the induction hypothesis, $\Pr_{z'\in S^{n-1}}[p_1(z') \neq 0] \geq \frac{d-d_1}{|S|}$. Let $h(Z_n) = p(z', Z_n)$. Given that $p_1(z') \neq 0$, $h(Z_n)$ is a degree d_1 polynomial and the event that $p(z) = 0$ is equivalent to the event that $h(z_n) = 0$ which happens with probability at most $\frac{d_1}{|S|}$. Combining the two above we find that the probability that $p(z) = 0$ is at most $d/|S|$. ▌

The Hadamard encoding is inspired by linear functions in one of two ways, both of which become important to us. On the one hand we may view $H(a)$ as defining a linear function $L(w_1, \ldots, w_k)$ (i.e., a homogeneous polynomial of degree 1 in k variables from $\mathbb{F}_2$), with $L(w) = H(a)(w)$. On the other hand we may view the Hadamard encoding as being indexed by all the linear functions $L : \{0,1\}^k \to \{0,1\}$ with $H(a)(L) = L(a)$.

The properties of the Hadamard encoding that make it useful in our transformation relate to its "local correctibility" and the "local testability".

Proposition 6.7 *If a string $X \in \{0,1\}^{2^k}$ indexed by linear functions satisfies $\delta(H(a), X) < \frac{1}{4}$, then a is uniquely defined by this property, and for every L, the probability over uniformly chosen L_1 that $H(a)(L) = X(L + L_1) - X(L_1)$ is at least $1 - 2\delta(H(a), X)$.*

Proof: The first assertion follows easily by contradiction. Suppose there exist $a \neq b$ such that $\delta(H(a), X), \delta(H(b), X) < \frac{1}{4}$. Viewing $H(a)$ as the evaluations of a linear function L_1 with $H(a)(w) = L_1(w)$ and letting L_2 denoting the corresponding function for $H(b)$, we have $L_1 - L_2$ is a non-zero polynomial of degree 1 over $\mathbb{F}_2$. By Lemma 6.6, we have that the probability that $L_1(w) - L_2(w) = 0$ is at most $1/2$ and thus $\delta(H(a), H(b)) \geq 1/2$. But then, by the triangle inequality, we have $\delta(H(a), H(b)) \leq \delta(H(a), X) + \delta(H(b), X) < \frac{1}{2}$ giving us the desired contradiction.

For the second part let $\delta = \delta(H(a), X)$. Note that the probability (over L_1) that $X(L+L_1) \neq H(a)(L+L_1)$ is at most δ since $L+L_1$ is uniformly distributed over all linear functions. Next note that the probability that $X(L_1) \neq H(a)(L_1)$ is also at most δ (though of course this is not independent of the previous event). The probability that either of these events occur is at most 2δ, and if neither events occur, then we have $X(L+L_1) - X(L_1) = H(a)(L+L_1) - H(a)(L_1) = H(L)$. ▌

Lemma 6.8 (Linearity testing, [16, 11]) *If a string $X \in \{0,1\}^{2^k}$ indexed by linear functions $L : \{0,1\}^k \to \{0,1\}$ satisfies $\Pr_{L_1,L_2}[X(L_1) + X(L_2) \neq X(L_1 + L_2)] \leq \delta$ then there exists a string a such that $\delta(H(a), X) \leq \delta$.*

We defer the proof of this lemma to Appendix A. Roughly this construction will replace a vertex u of G by a collection of vertices $\{u(L)\}_L$ where L ranges over all the linear function. It will also have many 3-uniform constraints on the coloring A' (which can be viewed as 4-uniform ones, by repeating one of the arguments) of the form $A'(u(L_1)) + A'(u(L_2)) = A'(u(L_1 + L_2))$. The lemma above says that if 90% of such constraints are satisfied by A', then it must be the case that there is an a such $H(a)(L)$ usually (for 90% of the L's) equals $u(L)$ and so it is reasonable to view A' as suggesting the coloring $A(u) = D(\langle A'(u(L))\rangle_L)$. Furthermore, for any fixed linear function L, we can effectively recover $H(A(u))(L)$ by looking at $A'(u(L+L_1)) - A'(u(L_1))$ for a randomly chosen L_1. This will allow us to create several other local constraints later. We now move to the constraint gadgets.

6.2 Enforcing quadratic constraints

Recall that our goal is to build "gadgets", i.e., a collection of 4-uniform constraints on a 2-coloring of some variables, that somehow verify that a 2-uniform constraint on a 2^k-coloring of two variables. Furthermore, we are assuming that the constraint is given by a collection of degree two polynomials on the $2k$ variables representing the colorings of the two variables. For concreteness let us fix a constraint c_e on the coloring of vertices u and v. Let $X_1, \ldots, X_k$ denote the Boolean variables representing the coloring of u and $Y_1, \ldots, Y_k$ represent the Boolean variables representing the coloring of v. Further, let $P_1(X,Y), \ldots, P_m(X,Y)$ denote the polynomials representing the constraint c_e (i.e., c_e is satisfied by the coloring a, b if $P_j(a,b) = 0$ for every $j \in [m]$).

It turns out that the Hadamard encoding brings us close to the goal of verifying such constraints. Indeed if the constraint were given by a *single, homogeneous, degree one* polynomial on $2k$ variables, then we would be done, given the Hadamard encoding gadget for the vertices u and v. To see this let $P_1(X,Y) = P_{11}(X) + P_{12}(Y)$ where $P_{11}(X)$ and $P_{12}(Y)$ are both linear. Now place constraints $A'(u(L_1)) + A'(u(L_2)) = A'(u(L_1 + L_2))$ for every pair of linear functions L_1, L_2, similar collection of constraints for v and then finally add constraints $A'(u(L_3 + P_{11})) - A'(u(L_3)) + A'(v(L_4 + P_{12}) - A'(v(L_4)) = 0$ for every pair of linear functions L_3 and L_4. If the decoding of a coloring A' leads to a coloring A that satisfies c_e, then we are set; else we find ourselves in one of two cases: Either $\langle A'(u(L))\rangle_L$ is not 0.1-close to some encoding $H(a)$ (or $\langle A'(v(L))\rangle_L$ is not 0.1-close to some encoding $H(b)$) in which case the 10% of the first class of constraints will reject; or both are0.1-close to encodings $H(a)$ and $H(b)$ with (a,b) not satisfying the constraint c_e, in which case $P_1(a,b) = P_{11}(a) + P_{12}(b) = 1$. But by Lemma 6.8, with probability at most0.2 we have $P_{11}(a) \neq A'(u(L_3 + P_{11})) - A'(u(L_3))$. Similarly the probability that have $P_{12}(b) \neq A'(v(L_4 + P_{12})) - A'(u(L_4))$. Thus with the remaining probability of0.6 we have $A'(u(L_3+P_{11}))-A'(u(L_3))+A'(v(L_4+P_{12}))-A'(u(L_4)) = P_1(a,b) = 1$ and so the latter test will reject. Thus we get an absolute constant fraction of the constraints in G' corresponding to c_e reject if the decoding of A' does not lead to a coloring satisfying c_e.

In what follows we see how to extend this idea to include a collection of degree 2 polynomials. To do so we extend the Hadamard encoding to now include the values of all degree two polynomials at some point a. We call this encoding the Quadratic encoding.

Definition 6.9 *The Quadratic Encoding Q maps strings of length k to strings of length $2^{\binom{k+1}{2}}$. Given a string $a \in \{0,1\}^k$ let $b \in \{0,1\}^{\binom{k+1}{2}}$, be the string indexed by pairs $i \leq j \in [k]$ with $b_{ij} = a_i a_j$. (Note that $b_{ii} = a_i^2 = a_i$.) The quadratic encoding $Q(a)$ is simply the Hadamard encoding of b.*

In the case of the quadratic function encoding it is possible to view the coordinates as corresponding to homogeneous polynomials of degree 2 and thus $Q(a) = \langle q(a) \rangle_q$ where q ranges over all degree two homogeneous polynomials on k variables.

Testing if a string $Y \in 2^{\binom{k+1}{2}}$ is close to the quadratic encoding of some a is a little more complex than testing Hadamard codes but not too much more so, given the Hadamard encoding X of a. We first test to ensure Y is close to the Hadamard encoding of some string b, and then we need to test if $b_{ij} = a_i a_j$ The main idea at this point is to consider a degree 2 homogeneous polynomial of the form $L_1(a)L_2(a)$ and check to see that X and Y agree with respect to this polynomial. In the case of X its value is given by $X(L_1)X(L_2)$. On the other hand Y's "opinion" for the value of this polynomial is given by a majority vote over quadratic polynomial q of the difference $Y(q + L_1L_2) - Y(q)$. Analyzing this test carefully (and we will do so in the coming subsections) shows that if Y is not close to the quadratic function encoding of a when X is close to the Hadamard encoding of a then a constant fraction of these constraints are not satisfied.

We remark that over $\mathbb{F}_2$, homogeneous polynomials essentially capture all interesting polynomials since for every degree two polynomial P we can associate a homogeneous degree 2 polynomial P_H and a constant $\beta \in \mathbb{F}_2$ such that for every $a \in \mathbb{F}_2^k$ we have $P(a) = P_H(a) + \beta$. Using this observation and the test described it is easy to extend our earlier "gadget" to the case where the constraint c_e is a single degree two polynomial. The only remaining hurdle is that we are dealing with a system of degree two polynomial; and we deal with this aspect next.

We now consider the case where c_e is the conjunction of polynomials $P_1, \ldots, P_m$ where m is arbitrary. Note that one obvious strategy is to repeat the exercise of testing if $P_j(a,b) = 0$ for every $j \in [m]$. But this could lead to situations where only a fraction $\frac{1}{m}$ of the constraints (corresponding a single unsatisfied P_j) being unsatisfied when c_e is not satisfied.

To avoid such scenarios we use once again the idea underlying the Hadamard code. Fix an coloring a, b to the pair u, v and let $z_j = P_j(a, b)$. To test if $z = \langle z_j \rangle_j = \vec{0}$ we could enumerate all the j's as above, or adopt the "error-correcting" strategy, where we check to see if the wth coordinate of $H(z)(w) = z \cdot w$ is zero for a randomly chosen index w. If $z = 0$ then certainly this is zero; on the other hand, if $z \neq 0$ then the probability that this happens is equal to $1/2$ (by the Schwartz-Zippel lemma). But now we note that $z \cdot w = \sum_{j=1}^{m} w_j z_j = \sum_{j=1}^{m} w_j P_j(a, b)$ is simply the evaluation of a single degree two polynomial at a, b a condition for which we already know how to build a gadget.

Finally, while the above shows how to integrate m polynomials into one without losing in the performance of the reduction, we still need to ensure that the size of the reduction (i.e., the number of constraints) is not too large. In turn this requires a bound on m. To get an upper bound, note that without loss of generality we can assume that the P_j's are linearly independent. (If there exists $\lambda_2, \ldots, \lambda_m \in \{0, 1\}$ such that $P_1 \equiv \sum_{j=2}^{m} \lambda_j P_j$, then it follows that $P_1(a, b) = 0$ is a redundant constraint given $P_j(a, b) = 0$ for $j \in \{2, \ldots, m\}$.) So we conclude that $m \leq \binom{k+2}{2}$.

This completes the motivation for our gadget and now we describe this gadget formally

6.3 The actual transformation

We now give the transformation from $\tilde{\mathcal{G}}_K$ where $K = 2^k$ to $G' \in \tilde{\mathcal{G}}_2^{(4)}$.

Fix an instance $G \in \tilde{\mathcal{G}}_K$. Let V denote the vertices of G and E its edges. For $e \in E$ incident on u and v, let c_e be the conjunction of degree two polynomials $P_{e,1}, \ldots, P_{e,m}$ on the $2k$ variables corresponding to the coloring of u and v.

We start by describing the vertex set V' of G'. V' comprises the following:

- A vertex $u(L)$ for every $u \in V$ and every linear function $L : \{0,1\}^k \to \{0,1\}$
- A vertex $e(L)$ for every $e \in E$ and every linear function $L : \{0,1\}^{2k} \to \{0,1\}$.
- A vertex $e(q)$ for every $e \in E$ and every homogeneous degree two polynomial $q : \{0,1\}^{2k} \to \{0,1\}$.

We now describe the (hyper)edges and constraints of G'. For every $e \in E$ incident to u and v with c_e being the conjunction of the polynomials $P_{e,1}, \ldots, P_{e,m}$ and for every pair of linear functions $L_1, L_2 : \{0,1\}^k \to \{0,1\}$, linear functions $L_3, L_4 : \{0,1\}^{2k} \to \{0,1\}$, every pair of homogeneous degree two polynomials $q_1, q_2 : \{0,1\}^{2k} \to \{0,1\}$, for every vector $w \in \{0,1\}^m$ and every index $i \in [7]$, we have an edge $e' = e(L_1, L_2, L_3, q_1, q_2, w, i)$ in E' where the vertices incident to e' and the constraint $c_{e'}$ depend on the value of i as follows:

$i = 1$: (Hadamard test for $u(L)$'s): e' is incident to $u(L_1), u(L_2)$ and $u(L_1 + L_2)$ and is satisfied if $A'(u(L_1)) + A'(u(L_2)) = A'(u(L_1 + L_2))$.

$i = 2$: (Hadamard test for $v(L)$'s): e' is incident to $v(L_1), v(L_2)$ and $v(L_1 + L_2)$ and is satisfied if $A'(v(L_1)) + A'(v(L_2)) = A'(v(L_1 + L_2))$.

$i = 3$**:** (Hadamard test for $e(L)$'s): e' is incident to $e(L_3), e(L_4)$ and $e(L_3+L_4)$ and is satisfied if $A'(v(L_3)) + A'(v(L_4)) = A'(v(L_3 + L_4))$.

$i = 4$**:** (Hadamard test for $e(q)$'s): e' is incident to $e(q_1), e(q_2)$ and $e(q_1 + q_2)$ and is satisfied if $A'(e(q_1)) + A'(e(q_2)) = A'(e(q_1 + q_2))$.

$i = 5$**:** (Consistency of $u(L)$'s, $v(L)$'s and $e(L)$'s): Let L_{12} denote the linear function given by $L_{12}(X, Y) = L_1(X) + L_2(Y)$. Then e' is incident to $u(L_1)$, $v(L_2)$, and $e(L_{12})$ and is satisfied if $A'(u(L_1)) + A'(v(L_2)) = A'(e(L_{12}))$.

$i = 6$**:** (Consistency of $e(L)$'s and $e(q)$'s): e' is incident to $e(L_3)$, $e(L_4)$, $e(q_1)$ and $e(q_1+L_3L_4)$ and is satisfied if $A'(e(L_3))A'(e(L_4)) = A'(e(q_1 + L_3L_4)) - A'(e(q_1))$.

$i = 7$**:** (Satisfaction of c_e): Let P_H be a homogeneous degree two polynomial and $\beta \in \{0, 1\}$ be such that for every $a, b \in \{0, 1\}^k$ $\sum_{j=1}^{m} w_j P_j(a, b) = P_H(a, b) + \beta$. e' is incident to vertices $e(q_1 + P_H)$ and $e(q_1)$ and is satisfied if $A'(e(q_1 + P_H)) - A'(e(q_1)) + \beta = 0$.

This completes the description of the gadget. While normally a gadget of such complexity would entail extremely complex analysis, in our case the analysis turns out to be quite streamlined as we see below.

6.4 Analysis

We start by taking care of the more obvious elements needed to show that the transformation from G to G' is $(\epsilon_3, 1)$-linear.

Proposition 6.10 *In the transformation of Section 6.3 from G to G', we have:*

1. *The size of G' is linear in the size of G. Specifically $|G'| \leq 2^{O(k^2)} \cdot |G|$.*
2. *The constraints of G' are degree two polynomials in at most 4 variables.*
3. *If G is satisfiable, then so in G'.*

Proof: For Part 1, we inspect the construction above. It is easy to see that the size of G' is only linearly large in the size of G since every vertex/constraint of G leads to $2^{O(k^2)}$ vertices/constraints in G'. (Note that this would be an enormous blowup even for $k = 10$; however it is a fixed constant for any constant k.)

For Part 2, note that each constraint depend on at most 4 variables, and each constraint of them expect some polynomial of degree one or two to be zero. Indeed except for the constraints corresponding to the indices $i = 6$, all the constraints are of degree one.

For Part 3, let $A : V \to \{0, 1\}^k$ be an assignment satisfying G. Then consider the following coloring A' to the vertices of G':

- $A'(u(L)) = H(A(u))(L)$.
- $A'(e(L)) = H((A(u), A(v)))(L)$, where e is incident to u and v; and $(A(u), A(v))$ is viewed as an element of $\{0,1\}^{2k}$.
- $A'(e(q)) = Q(A(u), A(v))(q)$ where $Q(\cdot)$ denotes the Quadratic Encoding function.

It can be verified by inspection that this assignment satisfies every constraint $c_{e'}$. ▌

We now move to the crucial part of the analysis, namely the analysis of the soundness of the transformation. The following lemma argues this by showing that if G' has a coloring which leaves only a small fraction of the clauses unsatisfied, then G has a coloring which also leaves a small fraction unsatisfied.

Lemma 6.11 *There exists a constant $\epsilon_3 > 0$ such that the following holds for every $\tau \geq 0$: Let A' be a coloring of the vertices of G' satisfying $1 - \epsilon_3 \cdot \tau$ fraction of constraints of G'. Let A be the coloring obtained by setting $A(u) = D(\langle A'(u(L)) \rangle_L)$, where L ranges over all linear functions from $\{0,1\}^k$ to $\{0,1\}$ and $D : \{0,1\}^{2^k} \to \{0,1\}^k$ denotes the Hadamard decoding function. Then A satisfies $1 - \tau$ fraction of the constraints of G.*

Proof: We prove the lemma for $\epsilon_3 = 0.001$. (Note that we don't attempt to optimize the constant here.)

For $e \in E$ let p_e denote the probability (over random choices of $L_1, L_2, L_3, L_4, q_1, q_2, w, i$ from the appropriate domains) that the constraint $e' = e(L_1, L_2, L_3, L_4, q_1, q_2, w, i)$ is *not* satisfied by the assignment A'. By an averaging argument, we note that the fraction of e's for which $p_e \geq \epsilon_3$ is at most τ. We show below that for every e such that $p_e < \epsilon_3$, the coloring A satisfies the constraint c_e. This will suffice to prove the lemma.

Fix e such that $p_e < \epsilon_3$. Let e be incident to u and v and let c_e be the conjunction of constraints $P_j(A(u), A(v)) = 0$ for $j \in [m]$. For $i \in [7]$, let $p_{e,i}$ denote the probability (now over the remaining parameters $L_1, L_2, L_3, L_4, q_1, q_2, w$) that $e' = e(L_1, L_2, L_3, L_4, q_1, q_2, w, i)$ is not satisfied by A'. Note that we have $p_{e,i} \leq 7\epsilon_3$ for every $i \in [7]$.

Linearity: Using the fact that $p_{e,1}$, $p_{e,2}$, $p_{e,3}$, and $p_{e,4}$ are all at most $7\epsilon_3 = 0.007 < 0.01$, we conclude (using Lemma 6.8) that there exists strings $a, b \in \{0,1\}^k$, $c \in \{0,1\}^{2k}$, and $d \in \{0,1\}^{\binom{2k+1}{2}}$ such that $\langle A'(u(L)) \rangle_L$ is .01-close to $H(a)$, $\langle A'(v(L)) \rangle_L$ is .01-close to $H(b)$, $\langle A'(e(L)) \rangle_L$ is .01-close to $H(c)$, and $\langle A'(e(q)) \rangle_q$ is .01-close to $H(d)$. Furthermore, by the definition of the Hadamard decoding and the uniqueness of this decoding (Proposition 6.7), we have that $A(u) = a$ and $A(v) = b$. It suffices to prove that $P_j(a, b) = 0$ for every $j \in [m]$ and we will do so below. This will involve relating a, b, c, d to each other and then using the constraints corresponding to $i = 7$.

Consistency of a, b, c: Suppose $c \neq (a, b)$. We will show that in this case the probability that the constraint $c_{e'}$ for $e' = e(L_1, L_2, L_3, L_4, q_1, q_2, w, 5)$ is unsatisfied is at least

.47 $> 7\epsilon_3$ which contradicts our bound on $p_{e,5}$. Recall that $c_{e'}$ is satisfied if $A'(u(L_1)) + A'(v(L_2)) = A'(e(L_{12}))$. Since $\langle A'(u(L)) \rangle_L$ is0.01 close to $H(a)$, we have that the probability (over L_1) that $A'(u(L_1)) \neq H(a)(L_1)$ is at most .1. Similarly we have that the probability (over L_2) that $A'(v(L_2)) \neq H(b)(L_2)$ is also at most .01. Finally since L_{12} is a random function chosen uniformly among all linear functions mapping $\{0,1\}^{2k}$ to $\{0,1\}$, we also have that the probability (over L_{12}) that $A'(e(L_{12})) \neq H(c)(L_{12})$ is at most .01. Finally we note that is $c \neq (a,b)$ then the probability (over L_{12}) that $H(c)(L_{12}) = H(a,b)(L_{12})$ is at most $\frac{1}{2}$. By the union bound the probability that at least one of the above events occurs is at most .53. Thus with probability at least .47 none of the events listed above occur and then we have

$$\begin{aligned} A'(u(L_1)) + A'(v(L_2)) &= H(a)(L_1) + H(b)(L_2) \\ &= H(a,b)(L_{12}) \\ &\neq H(c)(L_{12}) \\ &= A'(e(L_{12}) \end{aligned}$$

and the constraint e' rejects, as claimed. We conclude thus that $c = (a,b)$.

Consistency of c and d: Again suppose $d_{ij} \neq c_i c_j$ for some $i \leq j$. We now claim that the probability (essentially over L_3, L_4 and q_1) that the constraint $c_{e'}$, for

$$e' = e(L_1, L_2, L_3, L_4, q_1, q_2, w, 6),$$

is unsatisfied is at least $0.21 > 7\epsilon_3$ Recall that $c_{e'}$ is satisfied if $A'(e(L_3))A'(e(L_4)) = A'(e(q_1+L_3L_4)) - A'(e(q_1))$. As in the previous case, we conclude that the probability at least one of the events $A'(e(L_3)) \neq H(c)(L_3)$, $A'(e(L_4)) \neq H(c)(L_4)$, $A'(e(q_1+L_3L_4)) \neq H(d)(q_1 + L_3L_4)$, or $A'(e(q_1)) \neq H(d)(q_1)$ is at most .04. We show next that the probability that $H(d)(L_3L_4) = H(c)(L_3) \cdot H(c)(L_4)$ is at most 3/4. To see this define the polynomial $f(X_1, \ldots, X_{2k}, Y_1, \ldots, Y_{2k}) = \sum_{i \leq j}(d_{ij} - c_i c_j) X_i Y_j$. Let $L_3(Z_1, \ldots, Z_{2k}) = \sum_{i=1}^{2k} \alpha_i Z_i$ and $L_4(Z_1, \ldots, Z_{2k}) = \sum_{i=1}^{2k} \beta_i Z_i$. Then note that $H(d)(L_3L_4) = H(c)(L_3) \cdot H(c)(L_4)$ exactly when $f(\alpha, \beta) = 0$. We first note that f is not identically zero (since $d_{ij} \neq c_i c_j$ for some i, j). Viewing f as a polynomial in only the X's (with coefficients from the field of rational functions in Y) f is of degree 1. Hence by Lemma 6.6 we have that $f(\alpha, Y)$ is non-zero with probability at least $\frac{1}{2}$. Furthermore, if so then $f(\alpha, Y)$ is a degree 1 polynomial in Y and so (again using Lemma 6.6) $f(\alpha, \beta)$ is non-zero with probability at least 1/2. Thus putting the two together we get that $f(\alpha, \beta)$ is non-zero with probability at least 1/4 or equivalently, it is zero with probability at most 3/4. Thus with probability that any of the events listed above occur is at most .79. Thus the probability that none of the events occur is at least .21 and in such case we have

$$\begin{aligned} A'(e(L_3))A'(e(L_4)) &= H(c)(L_3)H(c)(L_4) \\ &\neq H(d)(L_3L_4) \\ &= H(d)(q_1 + L_3L_4) - H(d)(q_1) \\ &= A'(e(q_1 + L_3L_4)) - A'(e(q_1)) \end{aligned}$$

and the constraint $c_{e'}$ is unsatisfied. We conclude thus that $d_{ij} = c_i c_j$. As a consequence we have that for a random homogeneous degree two polynomial q, $A'(e(q)) \neq Q(c)(q)$ with probability at most .01.

Satisfaction of c_e We move to the last step of our analysis. Assume for contradiction that $P_j(a,b) = P_j(c) \neq 0$ for some $j \in [m]$. Consider a random constraint $e' = e(L_1, \ldots, w, 7)$. Note that $c_{e'}$ is satisfied iff $A'(e(q_1+P_H)) - A'(e(q_1)) + \beta = 0$ where P_H and β were chosen so that $\sum_j w_j P_j(c) = P_H(c) + \beta$. First note that the probability that $\sum_j w_j P_j(c) = 0$ is at most 1/2 (over the choice of w). We also have $A'(e(q_1)) \neq q_1(c)$ with probability at most .01 and $A'(e(q_1 + P_H)) \neq (q_1 + P_H)(c)$ is also at most .01. Thus the probability that none of the above events happen is at least .48, and in such case we have

$$\begin{aligned} A'(e(q_1 + P_H)) - A'(e(q_1)) + \beta &= (q_1 + P_H)(c) - (q_1)(c) + \beta \\ &= P_H(c) + \beta \\ &\neq 0 \end{aligned}$$

and the constraint $c_{e'}$ is unsatisfied.

Thus we have that the colors $a = A(u)$ and $b = A(v)$ satisfy the constraint c_e whenever $p_e < \epsilon_3$ and this holds for $1 - \tau$ fraction of the edges e as desired. ▌

Proof of Lemma 6.3: The lemma holds for ϵ_3 as given by Lemma 6.11. The transformation is the one given in Section 6.3. The fact that it is satisfiability-preserving and linear-size preserving follows from Proposition 6.10. Finally, Lemma 6.11 implies that if the unsatisfiability of the source instance G is at least τ, then the unsatisfiability of the transformed instance G' is at least $\epsilon_3 \cdot \tau$. ▌

7 Conclusion

We hope the reader finds the above description to be somewhat useful, and motivating when reading Dinur's new approach to construction 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 Sudan [15] 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?

Appendix

A Analysis of Linearity Testing

Proof of Lemma 6.8: Recall that we wish to that if a string $X \in \{0,1\}^K$ indexed by linear functions $L : \{0,1\}^k \to \{0,1\}$ satisfies $\Pr_{L_1,L_2}[X(L_1)+X(L_2) \neq X(L_1+L_2)] \leq \delta$ then there exists a string a such that $\delta(H(a),X) \leq \delta$, where $K = 2^k$.

For the proof it will be convenient to view all strings as elements of $\{-1,1\}^K \subseteq \mathbb{R}^K$, using the transformation $0 \mapsto 1$ and $1 \mapsto -1$. We will also use the inner product $\langle Y, Z\rangle = \mathbf{E}_L[Y(L)Z(L)]$. Note that since $Y(L)Z(L) = 1$ if they agree on the Lth coordinate and -1 when they disagree, we have $\langle Y, Z\rangle = \mathbf{E}_L[Y(L)Z(L)] = 1 - 2\delta(Y,Z)$. Thus our goal is to prove that there exists a string a such its encoding $H(a)$ (also viewed as an element of $\{-1,+1\}^K$) satisfies $\langle X, H(a)\rangle \geq 1 - 2\delta$. On the other hand, the hypothesis can be reinterpreted to be saying that $\mathbf{E}_{L_1,L_2}[X(L_1)X(L_2)X(L_1+L_2)] \geq 1 - 2\delta$.

The crux of the proof is the observation that the strings $H(a)$ form an orthonormal basis of $\mathbb{R}^K$. It is obvious that $\langle H(a), H(a)\rangle = 1$. On the other hand, we also have $\langle H(a), H(b)\rangle = 1 - 2\delta(H(a),H(b)) = 0$ (whenever $a \neq b$. Finally we have $K = 2^k$ distinct a's making the $H(a)$'s a basis. Thus we can express any string $X = \sum_a \hat{X}_a H(a)$ where $\hat{X}_a = \langle X, H(a)\rangle$ is the ath Fourier coefficient of X. By Parseval's identity, we have that $\sum_a \hat{X}_a^2 = \langle X, X\rangle = 1$, while our goal is to show that $\max_a \hat{X}_a \geq 1 - 2\delta$.

We now express the quantity $\mathbf{E}_{L_1,L_2}[X(L_1)X(L_2)X(L_1+L_2)]$ in terms of the Fourier coefficients. We have

$$\begin{aligned}
&\mathbf{E}_{L_1,L_2}[X(L_1)X(L_2)X(L_1+L_2)] \\
&= \mathbf{E}_{L_1,L_2}[(\sum_a \hat{X}_a H(a)(L_1))(\sum_b \hat{X}_b H(b)(L_2))(\sum_c \hat{X}_c H(c)(L_1+L_2))] \\
&= \sum_{a,b,c} \hat{X}_a \hat{X}_b \hat{X}_c \mathbf{E}_{L_1,L_2}[H(a)(L_1)H(b)(L_2)H(c)(L_1+L_2)] \\
&= \sum_{a,b,c} \hat{X}_a \hat{X}_b \hat{X}_c \mathbf{E}_{L_1}[H(a)(L_1)H(c)(L_1)]\mathbf{E}_{L_2}[H(b)(L_2)H(c)(L_2)] \\
&= \sum_{a,b,c} \hat{X}_a \hat{X}_b \hat{X}_c \langle H(a), H(c)\rangle\langle H(b), H(c)\rangle \\
&= \sum_c \hat{X}_c^3
\end{aligned}$$

where the last equality uses the fact that $\langle H(a), H(c)\rangle = 0$ if $a \neq c$ and is 1 if $a = c$ (and similarly with b and c).

Thus the hypothesis yields $\sum_c \hat{X}_c^3 \geq 1 - 2\delta$. But then we have $1 - 2\delta \leq \sum_c \hat{X}_c^3 \leq \max_a \hat{X}_a \sum_c \hat{X}_c^2 = \max_a \hat{X}_a$, yielding $\max_a \hat{X}_a \geq 1 - 2\delta$ as required. ▮

B Expanders, eigenvalues and random walks

The main goal of this section is to prove Proposition 5.11. A version of this proposition appears in the original paper of Dinur [19]. Our version differs only in that we work with walks on directed graphs. The proof involves algebraic properties of expander graphs, which we describe next.

Definition B.1 (Eigenvalues of a graph) *Let G be a d-regular graph with n vertices. The adjacency matrix of G, is the matrix $A(G) = (a_{vw})$ whose rows and columns are indexed by elements of $V(G)$ and a_{vw} is the number of edges of the form (v, w). Then, $A(G)$ is a symmetric $n \times n$ matrix, and has n real eigenvalues. The largest eigenvalue of $A(G)$ is d (corresponding to the all 1s eigenvector $\mathbf{1}$, and all eigenvalues are bounded by d in absolute value. We denote these eigenvalues by $d = \lambda_1(G), \lambda_2(G), \ldots, \lambda_n(G)$, where $d = \lambda_1(G) \geq \lambda_2(G) \geq \cdots \geq \lambda_n(G) \geq -d$. Then, the second eigenvalue value of G is defined to be*

$$\lambda(G) = \max\{|\lambda_2|, |\lambda_n|\}.$$

Also, there is an orthonormal basis $\{v_1, v_2, \ldots, v_n\}$ of $\mathbb{R}^n$, where v_i is an eigenvector of $A(G)$ corresponding to $\lambda_i(G)$.

Remark: If we assume that each vertex of the d-regular graph has at least $\frac{d}{2}$ self-loops, then all eigenvalues of G are non-negative, and the second eigenvalue of G is $\lambda_2(G)$.

Theorem B.2 (Eigenvalues versus expansion, Dodziuk [21], Alon and Milman [2], Alon [3 *Let G be an (η, d)-positive expander. Then,*

$$\lambda_2(G)^2 + \eta^2 \leq d^2.$$

Remark: The proof below is essentially the same as the one presented by Hoory, Linial and Wigderson [27] for showing that $\lambda_2(G) \leq d - \frac{\eta^2}{2d}$. The proof actually shows the stronger bound stated above. (Note that $\sqrt{d^2 - \eta^2} \leq d - \frac{\eta^2}{2d}$.)

Proof: We consider vectors in $\mathbb{R}^n$ indexed by elements of $V(G)$. The matrix $A(G)$ acts on these such vectors in the natural way. It will be convenient to talk of these vector as functions from $V(G)$ to $\mathbb{R}$. Let $\langle f, g\rangle = \sum_v f(v)g(v)$ and $\|f\|_2 = (\langle f, g\rangle)^{1/2}$.

Let $\mathbb{R}^+$ denote the set of non-negative real numbers. We say that a function $f : V(G) \to \mathbb{R}^+$ is proper if $|\{v : f(v) > 0\}| \leq \frac{|V(G)|}{2}$. We have two claims.

Claim B.3 *If f is a proper function, then $\eta^2\|f\|_2^4 \leq d^2\|f\|^4 - \langle f, A(G)f\rangle^2$.*

Claim B.4 *There exists a non-zero proper f such that $\langle f, Af\rangle \geq \lambda_2(G)\|f\|_2^2$.*

The theorem follows immediately by combining these claims.

Proof of Claim B.3: First we introduce some notation to simplify the presentation. Rename the vertices of G as $1, 2, \ldots, n$, such that

$$f(1) \geq f(2) \geq \cdots \geq f(s) > f(s+1) = f(s+2) = \cdots = f(n) = 0,$$

for some $s \leq \frac{n}{2}$. Recall that in our definition of a graph each edge e of the form (i, j) has a reverse edge e^- of the form (j, i). Let E^+ be the set of edges in E of the form (i, j) where $j \geq i$. Consider the expression

$$\begin{aligned}
\sum_{(i,j)\in E^+} f(i)^2 - f(j)^2 &= \sum_{(i,j)\in E^+} \sum_{k=i}^{j-1} f(i)^2 - f(i+1)^2 \\
&= \sum_{i=1}^{s} |E_{\{1,\ldots,i\}}| \left(f(i)^2 - f(i+1)^2\right) \\
&\geq \sum_{i=1}^{s} \eta i \cdot \left(f(i)^2 - f(i+1)^2\right) \\
&= \eta \sum_{i=1}^{s} f(i)^2 \\
&= \eta\|f\|_2^2. \qquad (9)
\end{aligned}$$

On the other hand,

$$\begin{aligned}
\sum_{(i,j)\in E^+} f(i)^2 - f(j)^2 &= \sum_{(i,j)\in E^+} (f(i) - f(j))(f(i) + f(j)) \\
&\leq \left(\sum_{(i,j)\in E^+} (f(i) - f(j))^2\right)^{1/2} \times \left(\sum_{(i,j)\in E^+} (f(i) + f(j))^2\right)^{1/2} \\
&= \left(\sum_{(i,j)\in E^+} f(i)^2 + f(j)^2 - 2f(i)f(j)\right)^{1/2}
\end{aligned}$$

$$\times \left(\sum_{(i,j)\in E^+} f(i)^2 + f(j)^2 + 2f(i)f(j) \right)^{1/2}$$

$$= \left(d\|f\|_2^2 - 2 \sum_{(i,j)\in E^+} f(i)f(j) \right)^{1/2} \times \left(d\|f\|_2^2 + 2 \sum_{(i,j)\in E^+} f(i)f(j) \right)^{1/2}$$

$$= \left(d^2\|f\|_2^4 - \langle f, A(G)f\rangle^2 \right)^{1/2}. \tag{10}$$

The claim follows by combining (9) and (10). (End of Claim B.3)

Proof of Claim B.4: Let g be an eigenvector corresponding to the second eigenvalue $\lambda_2 = \lambda_2(G)$. Since g is orthogonal to $\mathbf{1}$ (all ones, the eigenvector corresponding to the eigenvalue $\lambda_1 = d$), g takes both positive and negative values. We may assume that $|\{v : g(v) > 0\}| \le \frac{|V(G)|}{2}$ by, if necessary, replacing g by $-g$. Let $f : V(G) \to \mathbb{R}^+$ be defined by

$$f(v) = \begin{cases} g(v) & \text{if } g(v) > 0 \\ 0 & \text{otherwise} \end{cases}.$$

Then, for all $v \in V(G)$ such that $f(v) > 0$, we have

$$(Af)(v) \ge (Ag)(v) = \lambda_2 g(v) = \lambda_2 f(v).$$

Thus,

$$\langle f, (Af)(v)\rangle = \sum_{v\in V(G)} f(v)\cdot(Af)(v) \ge \sum_{v\in V(G)} \lambda_2 f(v)^2 = \lambda_2\|f\|_2^2.$$

(End of Claim B.4)

▌

Proof of Proposition 5.11: Consider the infinite walk, where the verifier does not stop. It is enough to show that for this walk, and for all $j \ge 2$,

$$\Pr[\mathbf{e}_j \in F \mid \mathbf{e}_1 \in F] \le \frac{|F|}{|E|} + \left(\frac{\lambda(G)}{d}\right)^{j-2},$$

where λ is the second eigenvalue of G. (The factor $\left(1-\frac{1}{t}\right)^{j-i}$ in our claim accounts for the fact that we stop the walk after each step with probability $\frac{1}{t}$.) We will establish a more general statement: for any two sets of (directed) edges F_1 and F_2,

$$\Pr[\mathbf{e}_j \in F_2 \mid \mathbf{e}_1 \in F_1] \le \frac{|F_2|}{|E|} + \left(\frac{\lambda}{d}\right)^{j-2}\sqrt{\frac{|F_2|}{|F_1|}}.$$

Let π_i be the distribution of the vertex $\mathbf{v}_i$ conditioned on the event $\mathbf{e}_1 \in F_1$. For $v \in V(G)$, let $D_1(v)$ be the number of edges in F_1 coming into v and let $D_2(v)$ be the number of edges of F_2 leaving v. Then, $\pi_1(v) = D_1(v)/|F_1|$, and

$$\Pr[\mathbf{e}_j \in F_2 \mid \mathbf{e}_1 \in F_1] = \frac{1}{d} \sum_{v \in V(G)} D_2(v)\pi_{j-1}(v) = \frac{1}{d}\langle D_2, \pi_{j-1}\rangle.$$

If M is the transition matrix corresponding to the walk, then

$$\pi_{j-1} = M^{j-2}\pi_1 = M^{j-2}\left(\frac{D_1}{|F_1|}\right).$$

We may write $\pi_1 = \frac{D_1}{|F_1|} = \frac{1}{n}\mathbf{1} + \frac{D_1'}{|F_1|}$, where D_1' is the orthogonal projection of D_1 on the subspace orthogonal to the eigenvector $\mathbf{1}$ ($n = |V(G)|$). Then, we have

$$\begin{aligned}
\Pr[\mathbf{e}_j \in F_2 \mid \mathbf{e}_1 \in F_1] &= \frac{1}{d}\langle D_2, \pi_{j-2}\rangle \\
&= \frac{1}{d}\langle D_2, M^{j-2}\pi_1\rangle \\
&= \frac{1}{d}\langle D_2, \frac{1}{n}\mathbf{1} + M^{j-2}\left(\frac{D_1'}{|F_1|}\right)\rangle \\
&= \frac{1}{d}\langle D_2, \frac{1}{n}\mathbf{1}\rangle + \frac{1}{d}\langle D_2, M^{j-2}\left(\frac{D_1'}{|F_1|}\right)\rangle \\
&\leq \frac{|F_2|}{|E|} + \frac{1}{d|F_1|} \cdot \|D_2\|_2 \cdot \|M^{j-2}D_1'\|_2 \\
&\leq \frac{|F_2|}{|E|} + \frac{1}{d|F_1|} \cdot \|D_2\|_2 \cdot \left(\frac{|\lambda|}{d}\right)^{j-2} \|D_1\|_2 \\
&\leq \frac{|F_2|}{|E|} + \left(\frac{|\lambda|}{d}\right)^{j-2} \cdot \frac{1}{d|F_1|} \cdot \sqrt{\sum_{v \in V(G)} D_2(v)^2} \cdot \sqrt{\sum_{v \in V(G)} D_1(v)^2} \\
&\leq \frac{|F_2|}{|E|} + \left(\frac{\lambda}{d}\right)^{j-2} \cdot \frac{1}{d|F_1|} \cdot \sqrt{d \sum_{v \in V(G)} D_2(v)} \cdot \sqrt{d \sum_{v \in V(G)} D_1(v)} \\
&\leq \frac{|F_2|}{|E|} + \left(\frac{\lambda}{d}\right)^{j-2} \cdot \sqrt{\frac{|F_2|}{|F_1|}}.
\end{aligned}$$

▮

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] N. Alon and V.D. Milman. λ_1, isoperimetric inequalities for graphs and superconcentrators. *Journal of Combinatorial Theory, Series A*, 38(1):73–88, 1985.

[3] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.

[4] Noga Alon and Joel Spencer. *The Probabilistic Method.* John Wiley and Sons, Inc., 1992.

[5] S. Arora and C. Lund. Hardness of approximations. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS, 1995.

[6] 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.

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

[8] 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.

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

[10] 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.

[11] Mihir Bellare, Don Coppersmith, Johan Håstad, Marcos Kiwi, and Madhu Sudan. Linearity testing over characteristic two. *IEEE Transactions on Information Theory*, 42(6):1781–1795, November 1996.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[17] 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.

[18] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd ACM Symposium Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971.

[19] 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/`.

[20] 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.

[21] J. Dodziuk. Difference equations, isoperimetric inequalities, and transience of certain random walks. *Transactions of the American Mathematical Society*, 284(2):787–794, 1984.

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

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

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

[25] Venkatesan Guruswami, Daniel Lewin, Madhu Sudan, and Luca Trevisan. A tight characterization of NP with 3-query PCPs. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 8–17, Palo Alto, California, 8-11 November 1998.

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

[27] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the AMS*, 43(4):(to appear), October 2006. Full version available from `http://www.math.ias.edu/ avi/BOOKS/expanderbookr1.pdf`.

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

[29] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations, (R. Miller, J. Thatcher eds.)*, pages 85–103, 1972.

[30] Leonid A. Levin. Universal search problems. *Problemy Peredachi Informatsii*, 9(3):265–266, 1973.

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

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

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

[34] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, October 1980.

[35] 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`.

[36] Richard E. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM '79, Lecture Notes in Computer Science*, volume 72, pages 216–225, 1979.