Probabilistically checkable proofs

Madhu Sudan*
MIT CSAIL, 32 Vassar Street, Cambridge, MA 02139, USA

ABSTRACT

Can a proof be checked without reading it? That certainly seems impossible, no matter how much reviewers of mathematical papers may wish for this. But theoretical computer science has shown that we can get very close to this objective! Namely random consistency checks could reveal errors in proofs, provided one is careful in choosing the format in which proofs should be written. In this article we explain this notion, constructions of such probabilistically checkable proofs, and why this is important to all of combinatorial optimization.

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; F.0 [**Theory of Computation**]: General

General Terms

Algorithms, Theory, Verification

Keywords

Randomness, Logic, Compuational Complexity, Combinatorial Optimization

1. INTRODUCTION

The task of verifying a mathematical proof is extremely onerous, and the problem is compounded when a reviewer really suspects a fatal error in the proof but still has to find an explicit one so as to convincingly reject a proof. Is there any way to simplify this task? Wouldn't it be great if it were possible to scan the proof cursorily (i.e., flip the pages randomly, reading a sentence here and a sentence there) and be confident that if the proof was buggy you would be able to find an error by such a superficial reading?

Alas, the current day formats of proofs don't allow such simple checks. It is possible to build a "proof" of any "assertion" (in particular ones that are not true) with just one single error, which is subtly hidden. Indeed, if you think back to the "proofs" of "1 = 2" that you may have seen in the past, they reveled in this flaw of current proof systems.

Fortunately this shortcoming of proofs is not an inherent flaw of logic. Over the past two decades, theoretical computer scientists have come up with novel formats for writing proofs of mathematical assertions. Associated with these formats are probabilistic algorithms that reviewers could (should?) use to verify the proofs. A reviewer using such a verification algorithm would be spared from reading the entire proof (and indeed will only read a "constant number of bits of the proof" - a notion we will elaborate on, and explain later). Any researcher who has a proof of a theorem can rewrite the proof in the prescribed format and offer it to the reviewer, with the assurance that the reviewer will be convinced of this new proof. On the other hand if someone claims a faulty assertion to be a theorem, and offers any proof (whether in the new format or not), the reviewer will discover an error with overwhelming probability.

In what follows we will attempt to formalize some of the notions that we have been using in a casual sense above. The central notion that will emerge is that of a "Probabilistically Checkable Proof (PCP)". Existence of such formats and verification algorithms often runs contrary to our intuition. Nevertheless they do exist, and in Section 4 we discuss two different approaches that have been used thus far to construct such PCPs.

PCPs are arguably fascinating objects. They offer a certain robustness to the logical process of verification that may not have been suspected before. While the process of proving and verifying theorems may seem of limited interest (say, only to mathematicians), we stress that the notion of a "convincing" argument/evidence applies much more broadly in all walks of life. PCPs introduce the fascinating possibility that in all such cases, the time taken to assess the validity of the evidence in supporting some claims, may be much smaller than the volume of the evidence. In addition to this philosophical interest in PCPs, there is a very different (and much more concrete) reason to study PCPs. It turns out that PCPs shed light in the area of combinatorial optimization. Unfortunately this light is "dark": The ability to construct PCPs mostly says that for many optimization problems where we knew that optimal solutions were (NP-

*Research supported in part by NSF Awards CCR-0726525 and CCR-0829672. The views expressed in this article are those of the author and not endorsed by NSF.

)hard to find, even near-optimal solutions are hard to find. We discuss these connections soon after we discuss the definitions of PCPs, in Section 3.

2. DEFINITIONS

We start by formalizing what we mean by "theorems", "'proofs", a "format" for proving them, and what it means to "change" such a format, i.e., what changes are allowed, and what qualities should be preserved. Hopefully, in the process we will also manage to establish some links between the topic of this article and computer science.

To answer these questions we go to back to the essentials of mathematical logic. A system of logic attempts to classify "assertions" based on their "truth value", i.e., separate true *theorems* from *false assertions*. In particular, theorems are those assertions that have "proofs". In such a system every sentence, including theorems, assertions, and proofs, is syntactically just a finite string of letters from a finite alphabet. (Without loss of generality the alphabet may be binary i.e., $\{0,1\}$.) The system of logic prescribes some axioms and some derivation rules. For an assertion to be true, it must be derivable from the axioms by applying a sequence of derivation rules. A *proof* of an assertion A may thus be a sequence of more and more complex assertions ending in the assertion A, with each intermediate assertion being accompanied with an explanation of how the assertion is derived from previous assertions, or from the axioms. The exact set of derivation rules used and the complexity of a single step of reasoning may vary from one logical system to another, but the intent is that eventually all logical systems (based on the same axioms) should preserve two essentials aspects: The set of theorems provable in any given system of logic should be the same as in any other. Furthermore, proofs should be "easy" to verify in each.

This final attempt to abstract the nature of a logical system leaves us with the question: What is "easy"? It is this aspect that led to the development of Turing and Church's work on the Turing machine. They ascribed "easiness" to being a mechanical process, as formalized by the actions of some Turing machine. Modern computational complexity is a little more careful with this concept. The task of verification of a proof should not only be "mechanical", but also "efficient", i.e., should be polynomial time computable. This leads to the following abstract notion of a system of logic: A system of logic is given by a polynomial time verification algorithm (or simply *verifier*) $V(\cdot,\cdot)$, that takes two inputs, an assertion A and some evidence E and produces a Boolean verdict "accept/reject". If $V(A,E) = \mathsf{accept}$ then and E is a proof of A. If A is an assertion such that there exists some E such that $V(A,E) = \mathsf{accept}$, then A is a theorem. In contrast to the notion that a proof is easy to verify, our current state of knowledge suggests that proofs may be hard to find, and this is the essence of the theory of NP-completeness [11, 27, 25]. Indeed the question "is NP=P?" is equivalent to the question "can every theorem be proved efficiently, in time polynomial in the length of its shortest proof?".

In what follows we will fix some system of logic, i.e., some verifier V_0 and consider other verifiers that are *equivalent* to this verifier. In such cases, when the set of theorems does not change, but the "proofs" may, we call the new system a "proof system". So, a proof system V would be equivalent to V_0 if the following conditions hold:

Completeness If A is a theorem (in V_0), then A is a theorem in V. Furthermore there is a proof of A in V that is at most a polynomial factor longer than its proof in V_0.

Soundness If A is not a theorem (in V_0) then A is not a theorem in V.

By allowing different verifiers, or proof systems, for the same system of logic, one encounters many different ways in which theorems can be proved. As an example, we show how the NP-completeness of the famous problem 3SAT allows one to produce formats for proofs that "localize" errors in erroneous proofs. Recall that an instance of 3SAT is a logical formula $\phi = C_1 \wedge \cdots \wedge C_m$ where C_j is the disjunction of 3 literals (variables or their complement). The NP-completeness of 3SAT implies the following: Given any assertion A and integer N, there is a 3CNF formula ϕ (of length bounded by a polynomial in N) such that ϕ is satisfiable if and only if A has a proof of length at most N (in V_0). Thus the deep logical question about the truth of A seems to reduce to a merely combinatorial question about the satisfiability of ϕ. The natural evidence for the satisfiability of ϕ would be an assignment and this is what we refer to as a "format" for proofs. The advantage of this format is that in order to reject an "erroneous proof", i.e., an assignment x that fails to satisfy ϕ, one only needs to point to one clause of ϕ that is not satisfied and thus only point to the three bits of the proof of x that this clause depends on to reveal the error. Thus errors are easily localized in this format.

Can one go further and even "find" this error efficiently? This is where probabilistically checkable proofs come in. In what follows we will attempt to describe verifiers that can verify proofs of satisfiability of 3CNF formulae (noticing that by the discussion above, this is as general as verifying proofs of any mathematical statement in any formal system).

2.1 Probabilistically checkable proofs

We start by formalizing the notion of the number of bits of a proof that are "read" by the verifier. In order to do so, we allow the verifier to have random access (oracle access) to a proof. So while the proof may be a binary string $\pi = \langle \pi[1] \ldots \pi[\ell] \rangle \in \{0,1\}^\ell$, the verifier gets to "read" the ith bit of π by querying an "oracle" for the ith bit of π and get $\pi[i]$ in response, and this counts as one *query*.

Probabilistically checkable proofs are motivated by the question: "how many bits of queries are really essential to gain some confidence into the correctness of a theorem"? It is easy to argue that if the verifier hopes to get away by querying only a constant number of bits of the proof, then it can not hope to be deterministic (else a constant time brute force search would find a proof that the verifier would accept). So we will allow the verifier to be probabilistic, and also it to make mistakes (with low probability). This leads us to the notion of a PCP verifier.

DEFINITION 2.1. *A* PCP verifier *of* query complexity $q(n)$, *and* gap $\epsilon(n)$ *is a probabilistic algorithm V that, given as input an assertion $A \in \{0,1\}^n$, picks a random string $R \in \{0,1\}^*$, makes oracle queries to a proof oracle $\pi : \{1,\ldots,\ell\} \to \{0,1\}$ and produces an accept/reject verdict,*

Running time *V always runs in time polynomial in n.*

Query complexity *V makes $q(n)$ queries into the proof π.*

Proof size *ℓ grows polynomial in n.*

Completeness *If A is a true assertion, then there exists a proof π that the verifier accepts on every random string R.*

Soundness, with gap $\epsilon(n)$ *If A is not true, then for every proof π, the probability, over the choice of the randomness R, that the verifier accepts at most $1 - \epsilon(n)$.*

The above definition associates two parameters to a PCP verifier, query complexity and gap. The query complexity is the number of bits of the proof that the verifier "reads". The gap is related to the "error" probability of the verifier, i.e., the probability of accepting false assertions. The larger the gap, the smaller the error. Since the definition above introduces several notions and parameters at once, let us use a couple of simple examples to see what is really going on.

The classical proof of satisfiability, takes a formula of length n, on upto n variables and gives a satisfying assignment. The classical verifier, who just reads the entire assignment and verifies that every clause is satisfied, is also a PCP verifier. Its query complexity $q(n)$ is thus equal to n. Since this verifier makes no error, its gap is given by $\epsilon(n) = 1$.

Now consider a probabilistic version of this verifier who chooses to verify just one randomly chosen clause of the given 3CNF formula. In this case the verifier only needs to query three bits of the proof and so we have $q(n) = 3$. How about the gap? Well, if a formula is not satisfiable, then at least one of the upto n clauses in the formula will be left unsatisfied by every assignment. Thus once we fix a proof π, the probability that the verifier rejects is least $1/n$, the probability with which the verifier happens to choose a clause that is not satisfied by the assignment π. This corresponds to a gap of $\epsilon(n) = 1/n$. (Unfortunately, there do exist (many) unsatisfiable 3CNF formulae which have assignments that may satisfy all but one clause of the formula. So the gap of the above verifier is really $\Theta(1/n)$.)

Thus PCP verifiers are just extensions of "classical verifiers" of proofs. Every classical verifier is a PCP verifier with high query complexity and no error (i.e., high gap), and can be converted into one with low (constant!) query complexity with high error (tiny gap). Indeed a smooth tradeoff between the parameters can also be achieved easily. To reduce the error (increase the gap) of a PCP verifier with query complexity q and gap ϵ, we could just run this verifier several, say k, times on a given proof, and reject if it ever finds an error. The new query complexity is now kq and if the theorem is not true then the probability of detecting an error is now $(1-\epsilon)^k$. The new error is approximately $1 - k\epsilon$ if $k \ll 1/\epsilon$ and thus the gap goes up by a factor of roughly k.

The fundamental question in PCP research was whether this tradeoff was essential, or was it possible to get high gap without increasing the number of queries so much. The PCP theorem states that such proof systems can be constructed!

THEOREM 2.2 (PCP THEOREM [4, 3, 12]). *3SAT has a PCP verifier of* constant *query complexity, and* constant *positive gap.*

There are two distinct proofs of this theorem in the literature and both are quite non-trivial. In Section 4 we will attempt to give some idea of the two proofs. But before that we will give a brief history of the evolution of the notion of a PCP, and one of the principal applications of PCPs in computer science. The exact constants (in the query complexity and gap) are by now well-studied and we will comment on them later in the concluding section.

2.2 History of definitions

The notion of a PCP verifier appears quite natural given the objective of "quick and dirty" verification of long proofs. However, historically, the notion did not evolve from such considerations. Rather the definition fell out as a byproduct of investigations in cryptography and computational complexity, where the notion of a PCP verifier was one of many different elegant notions of probabilistic verification procedures among interacting entities. Here we attempt to use the historic thread to highlight some of these notions (see [18] for a much more detailed look into these notions). We remark that in addition to leading to the definition of PCPs, the surrounding theory also influences the constructions of PCP verifiers — indeed one may say that one may have never realized that the PCP theorem may be true, had it not been for some of the prior explorations!

Interactive Proofs.

The first notion of a probabilistic proof system to emerge in the literature was that of an *interactive proof*. This emerged in the works of Goldwasser, Micali and Rackoff [20], and Babai and Moran [7]. An interactive proof consists of an interaction between two entities (or agents), a "prover" and a "verifier". The prover wishes to convince the verifier that some theorem A is true. The verifier is again probabilistic and runs in polynomial time, and should be convinced if the assertion is true and should reject any interaction with high probability if the assertion is not true.

The goal here was not to improve on the efficiency with which, say, proofs of 3-satisfiability could be checked. Instead the goal was to enhance the class of assertions that could be verified in polynomial time. A non-mathematical, day-to-day, example of an interactive proof would be that of distinguishing between two drinks. Imagine convincing your spouse or friend that buying an expensive bottle of wine, brand X, is really worth it. They may counter with a cheap bottle, brand Y, that they claim tastes exactly the same. You insist that they taste quite different, but it is hard to prove your point with any written proof. But this is something we could attempt to prove interactively, by a *blind taste* test. You can ask your spouse/friend to challenge you with a random glass of wine and if by tasting you can tell which brand it is, you manage to convince your partner that you may have a point — the two drinks do taste different. By repeating this test many times, you partner's conviction increases. Interactive proofs attempt to such capture phenomena and more. Indeed, this very example is converted to a very mathematical one by Goldreich, Micali and Wigderson [19], who use a mathematical version of the above to give proofs that two graphs are distinguishable, i.e., they are not isomorphic. (This is a problem for which we don't know how to give a polynomially long proof.)

The initial interest in Interactive Proofs came from two very different motivations. Goldwasser, Micali, and Rackoff were interested in the "knowledge" revealed in multiparty interactions, from the point of view of maintaining secrets. To understand this concept, they first needed to define interac-

tive protocols and interactive proofs; and then a formal measure of the knowledge complexity of this interaction. They noted that while interaction may reveal many bits of "information" (in the sense of Shannon [34]) to the interacting players, it may reveal little knowledge. For example, the interactive proof above that brand X is distinguishable from brand Y reveals no more "knowledge" than the bare fact that they are distinguishable. It doesn't, for example, tell you what features are present in one brand and not in the other, etc.

Babai and Moran's motivation was more oriented towards computational complexity of some number-theoretic and group-theoretic problems. They were able to present interactive proofs with just one rounds of interaction between verifier and the prover for a number of problems not known to be in NP (i.e,, not reducible to satisfiability). The implication, proved formally in later works, was that such problems may not be very hard computationally.

The theory of interactive proofs saw many interesting discoveries through the eighties, and then culminated in a surprising result in 1990 when Shamir [33], based on the work of Lund, Fortnow, Karloff and Nisan [28], showed the set of assertions that could be proved interactively were exactly those that could be verified by a polynomial space bounded verifier.

Multi-Prover- & Oracle-Interactive Proofs.

Part of the developments in the 1980s led to variations on the theme of interactive proofs. One such variation that became significant to the development of PCPs was the notion of a "Multi-prover Interactive Proof" (MIP) discovered by Ben-Or, Goldwasser, Kilian and Wigderson [8]. Ben-Or et al. were trying to replace some cryptographic assumptions (along the lines of statements such as "RSA is secure") in existing interactive proofs with non-cryptographic ones. This led them to propose the study of the setting where the proof comes from a pair of provers who, for the purpose of the verification task, are willing to be separated and quizzed by the verifier. The hope is that the verifier can quiz them on related facts to detect inconsistency on their part. This limits the prover's ability to cheat and Ben-Or et al. leveraged this to create protocols where they reveal little knowledge about the proof when trying to prove their assertion to the verifier. (Some of the information being leaked in single-prover protocols was occuring because the prover needed to prove its honesty, and this information leakage could now stop.)

Fortnow, Rompel and Sipser [16] tried to study the power of multiprover interactions when the number of provers increased from two to three and so on. They noticed that more than two provers doesn't enhance the complexity of the assertions that could be proved to a polynomial time verifier. Key to this discovery was the notion of an "Oracle-Interactive Proof". This is yet another variation in the theme of interactive proofs where the prover is "semi-honest". Specifically, the prover behaves as an oracle - it prepares a table of answers to every possible question that may be asked by the verifier and then honestly answers any sequence of questions from the verifier according to the prepared set of answers. (In particular, even if the history of questions suggests that a different answer to the latest question may increase the probability that the verifier accepts, the oracle does not change its mind at this stage.) Fortnow et al. noted that Oracle-Interactive-Proofs simulate any (polynomial) number of provers, and are in turn simulated by 2-prover proof systems with just one round of interaction (i.e., the verifier asks each of the two provers one question each, without waiting for any responses from the provers, and then the provers respond).

Subsequent to Shamir's result on the power of IP (single prover interactive proofs), Babai, Fortnow, and Lund [6] gave an analogous characterization of the power of MIP. They showed that the set of assertions that can be proved in polynomial time to a probabilistic verifier talking to two provers is the same as the set of assertions that could be verified in exponential time by a deterministic (classical) verifier. Thus the interaction with multiple provers reduced verification time from exponential to a polynomial!

Holographic proofs, PCPs.

In subsequent work, Babai, Fortnow, Levin and Szegedy [5] noticed that the notion of an oracle-interactive-proof was not very different from the classical notion of a proof. If one considers the table implied by the oracle prover and writes it down explicitly, one would get a very long string (potentially exponentially long in the running time of the verifier), which in effect was attempting to prove the assertion. In the result of [6], this oracle-based proof is not much longer than the classical proof (both have length exponential in the length of the assertion), but the oracle proof was much easier to check (could be checked in polynomial time). This led Babai et al. to name such proofs *holographic* (small pieces of the proof reveals its correctness/flaws). Babai et al. focussed on the computation time of the verifier and showed (in some careful model of verification) that every proof could be converted into a holographic one of slightly superlinear size, where the holographic one could be verified by the verifier in time that was some polynomial in the logarithm of the length of the proof.

Around the same time, with a very different motivation that we will discuss in the next section, Feige, Goldwasser, Lovasz, Safra, and Szegedy [15] implicitly proposed the concept of a *probabilistically checkable proof* with the emphasis now being on the query complexity of the verifier (as opposed to the computation time in holographic proofs). The notion of PCP was finally explicitly defined by Arora and Safra [4].

We stress that the theory of PCPs inherits much more than just the definition of PCPs from the theory of interactive proofs. The results, techniques, and even just the way of thinking, developed in the context of interactive proofs played a major role in the development of the PCP theorem. In particular, the notion of 2-player 1-round interactive proof and their equivalence to oracle-interactive proofs and hence PCPs plays a significant role in this theorem and we will use this notion to explain the proofs from a high-level in Section 4.

3. IMPLICATIONS TO COMBINATORIAL OPTIMIZATION

The notion of theorems and proofs have shed immense light on the complexity of combinatorial optimization. Consider a prototypical problem, namely graph coloring, i.e., the task of coloring the vertices of an undirected graph with the minimum number of possible colors so that the endpoints of every edge have distinct colors. The seminal works of Cook,

Levin, and Karp [11, 27, 25] show that this task is as hard as finding a proof of some given theorem. In other words, given an assertion A and estimate N on the length a proof, one can construct a graph G on $O(N^2)$ vertices with a bound K, such that G has a coloring with K or fewer colors if and only if A has a proof of length at most N. Furthermore, given a K-coloring of G, one can reconstruct a proof of A in time polynomial in N. Thus, unless we believe that proofs of theorems can be found in time polynomial in the length of the shortest proof (something that most mathematicians would find very surprising) we should also believe that graph coloring can not be solved in polynomial time.

Of course, graph coloring is just one example of a combinatorial optimization problem that was shown by the theory of NP-completeness to be as hard as the task of theorem-proving. Finding large independent sets in graphs, finding short tours for travelling salesmen, packing objects into a knapsack are all examples of problems for which the same evidence of hardness applies (see [17] for many more examples). The NP-completeness theory unified all these problems into the same one, equivalent to theorem proving.

Unfortunately, a somewhat more careful look into the different problems revealed many differences among them. This difference became apparent when one looked at their "approximability". Specifically, we say that an algorithm A solves a (cost) minimization problem Π to within some approximation factor $\alpha(n)$ if on every input x of length n, $A(x)$ outputs a solution whose cost is no more than $\alpha(n)$ factor larger than the minimum cost solution. For (profit) maximization problems, approximability is defined similarly: An $\alpha(n)$ approximation algorithm should produce a solution of cost at least the optimum divided by $\alpha(n)$. Thus $\alpha(n) \geq 1$ for every algorithm A and problem Π.

The NP-completeness theory says that for the optimization problems listed above find a 1-approximate solution (i.e., the optimum one) is as hard as theorem proving. However, for some NP-complete minimization problems, it may be possible to find a solution of cost, say, at most twice the optimum in polynomial time for every input. Indeed this happens for the travelling salesman problem on a metric space (a space where distances satisfy triangle inequality). If one finds a minimum cost spanning tree of the graph and performs a depth-first-traversal of this tree, one gets a "tour" that visits every node of the graph at least once and has a cost of at most twice the cost of the optimum travelling salesperson tour. (This tour may visit some vertices more than once, but such extra visits can be short-circuited. The short-circuiting only produces a smaller length tour, thanks to the triangle inequality.) Thus the travelling salesman problem with triangle inequalities (Δ-TSP) admits a polynomial time 2-approximation algorithm. Does this imply that every optimization problem admits a 2-approximation algorithm? Turns out that not even a $\sqrt{n}$-approximation algorithm is known for graph coloring. On the other hand the knapsack problem has a $(1+\epsilon)$-approximation algorithm for every positive ϵ, while the same is not known Δ-TSP. Thus while the theory of NP-completeness managed to unify the study of optimization problems, the theory of "approximability" managed to fragment the picture. Till 1990 however it was not generally known if the inability to find better approximation algorithms was for some inherent reason, or was it merely our lack of innovation. This is where the PCP theory came in to the rescue.

In their seminal work, Feige et al. [15], came up with a startling result. They showed that the existence of a PCP verifier as in the PCP Theorem (note that their work preceded the PCP Theorem in the form stated here, though weaker variants were known) implied that if the independent set size in a graph could be approximated to within any constant factor then NP would equal P! Given a PCP Verifier V and an assertion A, they constructed, in polynomial time, a graph $G = G_{V,A}$ with the property that every independent set in G corresponded to a potential "proof" of the truth of A, and the size of the independent set is proportional to the probability with which the verifer would accept that "proof". Thus if A were true, then there would be a large independent set in the graph of size, say K. On the other hand, if A were false, every independent set would be of size at most $(1-\epsilon)K$. Thus the gap in the acceptance probability of the verifier turned into a gap in the size of the independent set. A $1/(1-\epsilon/2)$-approximation algorithm would either return an independent set of size greater than $(1-\epsilon/2)K$, in which case A must be true, or an independent set of size less than $(1-\epsilon)K$ in which case we may conclude that A is false. Thus a $1/(1-\epsilon/2)$-approximation algorithm for independent sets suffices to get an algorithm to decide truth of assertions, which is an NP-complete task.

The natural next question is whether the connection between independent set approximation and PCPs is an isolated one — after all different problems do behave very differently with respect to their approximability, so there is no reason to believe that PCPs would also yield inapproximability results for other optimization problems. Fortunately, it turns out that PCPs do yield inapproximability results for many other optimization problems. The result of Feige et al. was followed shortly thereafter by that of Arora et al. [3] who showed that a broad collection of problems, there were non-trivial limits to the constant factor to which they were approximable, unless NP=P. (In other words, for each problem under consideration they gave a constant $\alpha > 1$ such that the existence of an α-factor approximation algorithm would imply NP=P.) This collection was the so-called MAX SNP-hard problems. The class MAX SNP had been discovered earlier by Papadimitriou and Yannakakis [30] and their work and subsequent works had shown that a varied collection of problems including the MAX CUT problem in graphs, Vertex Cover problem in graphs, Max 3SAT (an optimization version of 3SAT where the goal is to satisfy as many clauses as possible), Δ-TSP, Steiner trees in metric spaces, the shortest superstring problem were all MAX SNP-hard. Subsequently more problems were added to this set by Lund and Yannakakis [29] and Arora, Babai, Stern and Sweedyk [1]. The combined effect of these results was akin to that of Karp's work [25] in NP-completeness. They suggested that the theory of PCPs was as central to the study of approximability of optimization problems, as NP-completeness was to the exact solvability of optimization problems. Over the years there have been many successful results deriving inapproximability results from PCP machinery for a wide host of problems (see surveys by [2, 26] for further details). Indeed the PCP machinery ended up yielding not only a first cut at the approximability of many problems, but even very tight analyses in many cases. Some notable results here include the following:

- Håstad [22] showed that Max 3SAT does not have an α-approximation algorithm for $\alpha < 8/7$. This is tight

by a result of Karloff and Zwick [24] that gives an 8/7 approximation algorithm.

- Feige [14] gave a tight inapproximability result for the Set Cover problem.

- Håstad [21] shows that the clique size in n-vertex graphs can not be approximated to within a factor of $n^{1-\epsilon}$ for any positive ϵ.

Again, we refer the reader to some of the surveys for more inapproximability results [2, 26]. for further details).

4. CONSTRUCTIONS OF PCPS: A BIRD'S EYE VIEW

We now give a very high level view of the two contrasting approaches towards the proof of the PCP theorem. We stress that this is not meant to give insight, but rather a sense of how the proofs are structured. To understand the two approaches we find it useful to work with the notion of 2-prover one round proof systems. While the notion is the same as the one defined informally in Section 2, here we define the verifier more formally, and introduce the parameter corresponding to query complexity in this setting.

DEFINITION 4.1 (2IP VERIFIER, ANSWER SIZE, GAP). *A* 2IP verifier *of answer size $a(n)$, and gap $\epsilon(n)$ is a probabilistic algorithm V who, on input an assertion $A \in \{0,1\}^*$, picks a random string $R \in \{0,1\}^*$, makes one query each to two provers $P_1, P_2 : \{1,\ldots,\ell\} \to \{0,1\}^*$ and produces an* accept/reject *verdict, denoted $V^{P_1,P_2}(A;R)$, with the following restrictions, when $A \in \{0,1\}^n$:*

Running time *V always runs in time polynomial in n.*

Answer size *The prover's answers are each $a(n)$ bits long.*

Prover length *The questions to the provers are in the range $\{1,\ldots,\ell(n)\}$ where $\ell(n)$ is a polynomial in n.*

Completeness *If A is a true assertion, there exist provers P_1, P_2 such that $V^{P_1,P_2}(A;R)$ always accepts.*

Soundness, with gap $\epsilon(n)$ *If A is not true, then for every pair of provers P_1, P_2 the probability, over the choice of the randomness R, that $V^{P_1,P_2}(A;R)$ outputs* accept *is at most $1-\epsilon(n)$.*

The definition is (intentionally) very close to that of a PCP verifier, so let's notice the differences. Rather than one proof oracle, we now have two provers. But each is asked only one question, so effectively they are oracles! In PCPs, the response to a query is one bit long, but now the responses are $a(n)$ bits long. On the other hand in PCPs, the verifier is allowed to make $q(n)$ queries to the proof oracle, while here the verifier is only allowed one query each. Nevertheless PCP verifiers and MIP verifiers are closely related. In particular, the following proposition is really simple from the definitions.

PROPOSITION 4.2. *If 3SAT has a 2IP verifier of answer size $a(n)$ and gap $\epsilon(n)$, then 3SAT has a PCP verifier with query complexity $2 \cdot a(n)$ and gap $\epsilon(n)$.*

The PCP verifier simply simulates the 2IP verifier, with each query to the provers being simulated by $a(n)$ queries to the proof oracle.

Thus to prove the PCP theorem it suffices to give a 2IP verifier with constant gap and constant answer size. We start with the approach of Arora et al. [4].

4.1 Reducing answer size

The initial proofs of the PCP theorem approached their goal by holding the gap to be a constant, while allowing the 2IP verifier to have (somewhat) long answer sizes. Key to this approach were some alphabet reduction lemmas initiated in the work of [4]. Here we state two from [3], that suffice for our purposes.

LEMMA 4.3 ([3]). *There exists a constant $\delta > 0$ such that if 3SAT has a 2IP verifier with answer size $a(n)$ and gap ϵ, then 3SAT also has a 2IP verifier with answer size $(\log a(n))^2$ and gap $\epsilon \cdot \delta$.*

LEMMA 4.4 ([3]). *There exist constants $c < \infty$ and $\tau > 0$ such that if 3SAT has a 2IP verifier with answer size $a(n) = o(\log n)$ and gap ϵ, then 3SAT also has a 2IP verifier with answer size c and gap $\epsilon \cdot \tau$.*

Both lemmas above offer (pretty severe) reductions in answer sizes. Below, we show how they suffice to get the PCP theorem. Of course, the technical complexity is all hidden in the proofs of the two lemmas, which we will not be able to present. We simply mention that these lemmas are obtained by revisiting several popular "algebraic error-correcting codes" and showing that they admit query efficient probabilistic algorithms for "error-detection" and "error-correction". The reader is referred to the original papers [4, 3] for further details.

PROOF OF THEOREM 2.2. We start by noting that the classical (deterministic) verifier for 3SAT is also a 2IP verifier with answer size n and gap 1. Applying Lemma 4.3 we then get it thus has a 2IP verifier with answer size $(\log n)^2$ and gap δ. Applying Lemma 4.3 again we now see that is also has a 2IP verifier with answer size $(\log(\log n))^2$ and gap δ^2. Since $a(n) = o(\log n)$ we can now apply Lemma 4.4 to see that it has a 2IP verifier with answer size c and gap $\delta^2 \cdot \tau$. By Proposition 4.2 we conclude that 3SAT has a PCP verifier with query complexity $2c$ and gap $\delta^2\tau$. □

4.2 Amplifying error

We now turn to the new, arguably simpler, proof due to Dinur [12] of the PCP theorem. Since we are hiding most of the details behind some of the technical lemmas, we won't be able to completely clarify the simplicity of Dinur's approach. However we will be able to at least show how it differs right from the top level.

Dinur's approach to the PCP theorem is an iterative one. and rather than working with large answer sizes, this proof works with small gaps (during intermediate stages).

The approach fixes a "generalized graph k-coloring" problem as the problem of interest and fixes a canonical 2IP verifier for this problem. It starts by observing that 3SAT can be transformed to this generalized graph 3-coloring problem. It then iteratively transforms this graph into a different one, each time increasing the "gap of the instance". The final instance ends up being one that where the canonical

2IP verifier either accepts with probability 1, or rejects with constant probability (depending on whether the original instance is satisfiable or not), which is sufficient for the PCP theorem. We go into some more details of this approach below, before getting into the heart of the process which is the single iteration.

A *generalized graph k-coloring* problem has as an instance a graph $G = (V, E)$ and constraint functions $\pi_e : \{1, \ldots, k\} \times \{1, \ldots, k\} \to \{\mathsf{accept}, \mathsf{reject}\}$ for every edge $e \in E$. The *canonical 2IP verifier* for in instance expects as provers two oracles giving $\chi_1, \chi_2 : V \to \{1, \ldots, k\}$ and does one of the following: With probability $1/2$ it picks a random edge $e = (u, v) \in E$ queries for $\chi_1(u)$ and $\chi_2(v)$ and accepts iff $\pi_e(\chi_1(u), \chi_2(v)) = \mathsf{accept}$. With probability $1/2$ it picks a random vertex $u \in V$ and queries for $\chi_1(u)$ and $\chi_2(u)$ and accepts if and only if $\chi_1(u) = \chi_2(v)$. Note that the canonical 2IP verifier has answer size $\lceil \log_2 k \rceil$. An instance is *satisfiable* if the canonical 2IP verifier accepts with probability one. An instance is *ϵ-unsatisfiable* if the probability that the verifier rejects is at least ϵ.

Key to Dinur's iterations are transformations among generalized graph coloring problems that play with the gap and the answer size (i.e., # of colors allowed) of the 2IP verifiers. Since these transformations are applied many times to some fixed starting instance it is important that the transformations do not increase the problem size by much and Dinur insists that they only increase them by a linear factor. We define this notion formally below.

Definition 4.5. *A transformation T that maps instances of generalized graph k-coloring to generalized graph K-coloring is a $(k, K, \beta, \epsilon_0)$-linear-transformation if it satisfies the following properties:*

- *$T(G)$ has size linear in the size of G.*
- *$T(G)$ is satisfiable if G is satisfiable.*
- *$T(G)$ is $\min\{\beta \cdot \epsilon, \epsilon_0\}$-unsatisfiable if G is ϵ-unsatisfiable.*

Note that the parameter β above may be greater than 1 or smaller; and the effect in the two cases is quite different. If $\beta > 1$ then the transformation increases the gap, while if $\beta < 1$ then the transformation reduces the gap. As we will see below, Dinur's internal lemmas play with effects of both kinds (combining them in a very clever way).

The key lemma in Dinur's iterations do not play with the answer size and simply increase the gap and may be stated as below.

Lemma 4.6 (Gap Amplification Lemma). *There exists a constant $\epsilon_0 > 0$ such that there exists a polynomial-time computable $(3, 3, 2, \epsilon_0)$-linear-transformation T.*

Before getting a little into the details of the proof of this lemma, let us note that it suffices to prove the PCP theorem.

Proof of Theorem 2.2. We describe a 2IP verifier for 3SAT. The verifier acts as follows. Given a 3CNF formula ϕ of length n, it first applies the standard reduction from 3SAT to 3-coloring to get an instance G_0 of (generalized) graph 3-coloring which is 3-colorable iff ϕ is satisfiable. Note that this instance is $1/m$-unsatisfiable for some $m = O(n)$. The verifier then iteratively applies the transformation T to G_0 $\ell = \log m$ times. I.e., it sets $G_i = T(G_{i-1})$ for $i = 1, \ldots, \ell$. Finally it simulates the canonical 2IP verifier on input G_ℓ.

If ϕ is satisfiable, then so is G_i for every i, and so the canonical 2IP verifier accepts with probability 1. If ϕ is unsatisfiable then G_i is $\min\{2^i \cdot 1/m, \epsilon_0\}$-unsatisfiable and so G_ℓ is ϵ_0-unsatisfiable. Finally note that since each iteration increases the size of G_i only by a constant factor, the final graph G_ℓ is only polynomially larger than ϕ, and the entire process only requires polynomial time.

Note that the 2IP verifier thus constructed has answer size $\lceil \log_2 3 \rceil = 2$ bits. Its gap is ϵ_0. The conversion to a PCP verifier leads to one that has query complexity of 4 bits and gap $\epsilon_0 > 0$. □

We now turn to the magical gap-amplifying lemma above. Dinur achieves this lemma with two sub-lemmas, where the game between answer size and gap becomes clear.

Lemma 4.7 (Gap-Increase). *For every $k, \beta_1 < \infty$, there exists a constant $K < \infty$ and $\epsilon_1 > 0$ such that a $(k, K, \beta_1, \epsilon_1)$-linear-transformation T_1 exists.*

Note that a large $\beta_1 \gg 1$ implies the transformation enhances the unsatisfiability of an instance. The Gap-Increase lemma is claiming that one can enhance this unsatisfiability by any constant factor for an appropriate price in the answer size. The next lemma trades off in the other direction, but with a clever and critical switch of quantifiers.

Lemma 4.8 (Answer-Reduction). *For every k there exists a constant $\beta_2 > 0$ such that for every $K < \infty$ a $(K, k, \beta_2, 1)$-linear-transformation T_2 exists.*

The constant β_2 obtained from the lemma is quite small (very close to 0). But for this price in gap-reduction we can go from large answer sizes to small ones, and the price we pay in the unsatisfiability is independent of K! This allows us to combine the two lemmas to get the powerful gap amplification lemma as follows.

Proof of Lemma 4.6. Fix $k = 3$. Let β_2 and T_2 be as in Lemma 4.8. Apply Lemma 4.7 with $\beta_1 = 2/\beta_2$ and let K, ϵ_1 and T_1 be as guaranteed by Lemma 4.7. Let $T(G) = T_2(T_1(G))$. Then, it can be verified that T is a $(k, k, 2, \beta_2 \cdot \epsilon_1)$-linear-transformation. □

Finally we comments on the proofs of Lemmas 4.7 and 4.8. We start with the latter. The crux here is the independence of β_2 and K. A reader who attempts to use standard reductions, from say k-coloring to K-coloring would realize that this is non-trivial to achieve. But if one were to ignore the linear-size restriction, the PCP literature already gave such transformations before. In particular Lemma 4.4 above gives such a transformation provided $K = 2^{o(\log n)}$. When specialized to the case $K = O(1)$ the reduction also turns out to be a linear one.

Lemma 4.7 is totally novel in Dinur's work. To get a sense of this lemma, let us note that its principal goal is to reduce the error of the 2IP verifier and so is related to the standard question in the context of randomized algorithms: that of error-reduction. In the context of randomized algorithms this is well-studied. If one starts with any randomized algorithm to compute some function and say it produces the right answer with probability $2/3$ (and errs with probability $1/3$), then one can reduce the error by running this algorithm many times and outputting the most commonly seen answer. Repetition m times reduces the error to $2^{-\Omega(m)}$.

One could view a 2IP verifier as just another randomized procedure and attempt to repeat the actions of the verifier m times to reduce its error. This leads to two problems. First the natural approach increases the number of rounds of communication between the verifier and the prover to m-rounds and this is not allowed (by our definitions, which were crucial to the complementary lemma). A less natural, and somewhat optimistic, approach would be to repeat the random coin tosses 2IP verifier m times, collect all the questions that it would like to ask the, say, first prover and send them together in one batch (and similarly with the second prover). Analysis of such parallel repetition of 2IP verifiers was known to be a non-trivial problem [16, 32], yet even such an analysis would only solve the first of the problems with the "naive" approach to error-reduction. The second problem is that the transformation does not keep the size of the transformed instance linear in size and this turns out to be a fatal. Dinur manages to overcome this barrier by borrowing ideas from "recycling" of randomness [10, 23], which suggests approaches for saving on this blowup of the instance size. Analyzing these approaches is non-trivial, but Dinur manages to do so, with a relatively clean (and even reasonably short) proof. The reader is pointed to the original paper [12] for full details, and to a more detailed survey [31] for information on the context.

5. CONCLUSIONS

The goal of this writeup was to mainly highlight the notion of a probabilistically checkable proof, and its utility in computational complexity. Due to limitations on space and time, we were barely able to scratch the surface. In particular we did not focus on the explicit parameters and the tightest results known. The literature on PCPs is rich with a diversity of parameters, but we chose to focus on only two: The query complexity and the gap. The tradeoff between the two is already interesting to study and we mention one tight version, which is extremely useful in "inapproximability" results. Håstad [22] shows that the query complexity in the PCP theorem can be reduced to 3 bits, while achieving a gap arbitrarily close to 1/2. So a verifier confronted with a fallacious assertion can read just 3 bits of the proof, and would find an error with probability (almost) one-half!

One of the somewhat strange aspects of PCP research has been that even though the existence of PCPs seems to be a "positive" statement (verification can be very efficient), its use is mostly negative (to rule out approximation algorithms). One may wonder why the positive aspect has not found a use. We suggest that positive uses might emerge as more and more of our life turns digital, and we start worrying not only about the integrity of the data, but some of the properties they satisfy, i.e., we may not only wish to store some sequence of bits x, but also preserve the information that $P(x) = y$ for some program P that took x as an input.

One barrier to such uses is the current size of PCPs. PCP proofs, even though they are only polynomially larger than classical proofs; they are much larger, and this can be a prohibitive cost in practice. The good news is that this parameter is also improving. An optimistic estimate of the size of the PCP proof in the work of [22] might be around n^{10^6}, where n is the size of the classical proof! But recent results have improved this dramatically since and current best proofs [9, 12] work with PCPs of size around $O(n(\log n)^{O(1)})$ (so the constant in the exponent has dropped from 10^6 to $1 + o(1)$. Thus far this reduction in PCP size has come at the price of increased query complexity, but this concern is being looked into by current research and so a positive use of PCPs may well be seen in the near future.

Acknowledgments

I would like to thank the anonymous reviewers for their valuable comments, and for detecting (mathematical!) errors in the earlier version of this manuscript (despite the fact that this article is not written in the probabilistically checkable proof format).

6. REFERENCES

[1] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes and systems of linear equations. *Journal of Computer and System Sciences*, 54(2):317–331, April 1997.

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

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

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

[5] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd ACM Symposium on the Theory of Computing*, pages 21–32, New York, 1991. ACM Press.

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

[7] L. Babai and S. 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.

[8] 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.

[9] E. Ben-Sasson and M. 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, New York, 2005. ACM Press.

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

[11] S. 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.

[12] I. Dinur. The PCP theorem by gap amplification. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 241–250, New York, 2006.

ACM Press. Preliminary version appeared as an ECCC Technical Report TR05-046.

[13] I. Dinur and O. 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, Loc Alamitos, CA, USA, 2004. IEEE Press.

[14] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[15] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996.

[16] L. Fortnow, J. Rompel, and M. Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, 1994.

[17] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.

[18] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, volume 17 of *Algorithms and Combinatorics*. Springer-Verlag, 1998.

[19] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, July 1991. Preliminary version in *IEEE FOCS*, 1986.

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

[21] J. Håstad. Clique is hard to approximate within n to the power 1-epsilon. *Acta Mathematica*, 182:105–142, 1999.

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

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

[24] H. Karloff and U. Zwick. A 7/8-approximation algorithm for max 3sat? In *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97)*, pages 406–415, Washington, DC, USA, 1997. IEEE Computer Society.

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

[26] S. Khot. Guest column: inapproximability results via long code based pcps. *SIGACT News*, 36(2):25–42, 2005.

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

[28] C. Lund, L. Fortnow, H. J. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, October 1992.

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

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

[31] J. Radhakrishnan and M. Sudan. On DinurŠs proof of the PCP theorem. *Bulletin (New Series) of the American Mathematical Society*, 44(1):19–61, January 2007.

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

[33] A. Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992.

[34] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.