Proofs in Three Bits or Less

Jacob Denson

November 24rd, 2017

Our story starts with a 21st centry problem. Proofs have been getting far too long lately. Displayed below is a graph of the long proofs over history.

- 1799: The Abel-Ruffini theorem on the insolvability of the quintic is proved. 500 Pages.
- 1890: Killing's classification of simple complex Lie algebras, 180 pages.
- 1894: A rule and compass construction of a polygon with 65537 sides, 200 pages.
- 1963: The odd order theorem is proven, 255 pages.
- 1974: Computer proof of 4 Color theorem, 139 pages + calculations.
- 1983: General Form of Selberg trace formula, 1322 pages.
- 2004: Classification of finite simple groups, 100000-200000 pages.
- 2005: Kepler's conjecture on sphere packing (computer), hundreds of pages of arguments, plus GIGABYTES of computer calculations.
- 2007: Proof that a perfect game of checkers ends in a draw. YEARS OF CALCULATION, Maybe 250GB?
- 2014: Particular case of Erdös discrepency conjecture: Original proof 13 Gigabytes, 'shortened to' 850 Megabytes.
- 2016: Boolean pythagorean tribes problem requires 200 TERABYTES of data.
- 2017: 5th Schur's Number is 161, 2 Petabytes of Data.

In the 70s computer proofs started appearing, whose proofs were literally infeasible to check. In a combinatorics talk I was at on Tuesday it was off handedly mentioned that the talk was based off of a 600 page thesis! What can we do to stop this madness! One solution I can think of is to limit the size of a proof. Think of a kind of mathematical social media website where mathematicians post proofs, but these proofs are limited to 140 characters or less. Then the

proofs will be accessible to every mathematician, and even to the general public as well! And it's not like we'll run out of proofs, because there are about 26^{140} different sequences of characters we can make in 140 characters or less, and at least some portion of this sequence will form mathematical proofs. We'll be busy for a while! But while thinking about these dumb ideas, a colleague introduced me to a radical, actually intelligent way of viewing arguments emerging from cryptography, known as **interactive proofs**. The idea is kind of like the party game 20 questions. Suppose we wish to determine whether a given statement is true or false by asking questions to a given person, known as a **prover**. Whereas classical proofs are static, in interactive proofs we ask a set of yes/no questions to the prover, and then determine whether the prover really has a proof of the result. After a fixed number of questions, we must decide whether we are convinced by the answers we recieved. If the number of questions is small, this makes proofs very comprehendable!

An problem you might not realize is that the prover might make errors, or more maliciously, might have lied in response to your questions. Thus we must ask questions which are 'checkable'. An interesting question to ask is how many questions we must ask to become convinced of a particular mathematical statement? By convinced, we mean

- **Perfect Completeness**: If a statement is true, there exists a series of answers to our questions which will convince us the statement is correct.
- **Perfect Soundness**: If a statement is false, then *every* series of answers to our question will fail to convince us the statment is true.

If we choose our questions randomly, we can introduce weaker conditions:

- **Imperfect Completeness**: If a statement is true, then there exists a series of answers to our randomly chosen questions which will convince us the statement is correct with probability greater than $1/2$.
- **Imperfect Soundness**: If a statement is false, then *every* series of answers to our question will fail to convince us with probability greater than $1 - \varepsilon$.

Classical proofs are just interactive proofs that are perfectly complete and sound. The cost is inefficiency; For instance, the theorem

" This statement cannot be proven in less than a googol symbols "

can be correctly formalized in formal arithmetic, and cannot be proven in less than googolplex symbols. Thus incredibly short statements can have arbitrarily long proofs. However, if we are willing to avoid perfection, then there is a magical result from theoretical computing science.

Theorem 1 (The PCP Theorem)**.** *There exists a universal constant K such that, if a theorem is 'polynomial time checkable' (we say the theorem is 'in* **NP***'), then the theorem is checkable with perfect soundness, and imperfect completeness in K questions.*

If we are willing to have imperfect soundness, so we can be fooled with some neglible probability, we obtain an even better version, due to Håstad:

Theorem 2. *For $\varepsilon > 0$, every 'feasibly checkable' theorem is checkable with imperfect soundness and imperfect completeness in only* 3 *questions!*

In essense, this means that we can prove anything with only three bits! By a feasibly checkable theorem, we mean a problem such that a proof can be verified or rejected in a polynomial number of computational steps. For instance, the problem of checking whether a graph is three colorable is very difficult, and unless $\mathbf{P} = \mathbf{NP}$, we cannot check this in polynomial time. However, if someone gives us a *particular* three coloring, we can just check whether the coloring is valid by checking each edge, so the computation takes $|E|$ steps, which is polynomial in the size of the graph. This is a computationally feasible problem. Similarily, it is not obvious that we can check whether a given integer has exactly 100 prime factors easily (Remark: it is actually an open problem whether this is easy or not, to all you aspiring number theorists, with a state of the art solution which runs in less than

$$O\left(\exp \sqrt[3]{\frac{64}{9} n (\log n)^2}\right)$$

steps, where n is the number of bits required to specify the number to factorize. However, if you give me the prime factorization of the integer, then I can make sure you've given me exactly 100 prime factors, and that these factors multiply to equal the integer. Multiplication of two n bit numbers takes $O(n^2)$ steps, so, if our initial number is expressible in n bits, then we can first check each prime factor has less than n bits, in $O(n)$ steps, multiply all these integers in 100 $O(n^2) = O(n^2)$ steps, and then check whether the product we obtain is equal to the original number, so we can check a given factorization in $O(n^2)$ time. These are the family of problems that are 'feasibly checkable'.

The PCP theorem doesn't say that, in 3 questions, we can obtain a proof of the theorem, but instead that we can verify whether the prover *has* a proof of the theorem. For instance, the PCP theorem says that we can determine whether an prover has a decomposition of an integer into 100 prime factors by asking it 3 questions, but we might not actually know what this decomposition is. Note that asking the prover "Do you have a decomposition of the integer into 100 prime factors" is *not* a useful question, because the prover might have made a mistake, or lie, and there is no way to check this question is true.

Example. *Here is a practical example of the PCP theorem, which has been proposed for scaling the utility of certain digital currencies, like Bitcoin. One problem with digital, non centralized currencies is that there is no central authority deciding when fraud is taking place. The big realization of decentralized currency is that if we have a whole bunch of people volunteering to maintain the ledger of all transactions (in exchange for earning some bitcoins, this is mining), no person can individually break the ledger, because their ledger would disagree*

with everyone elses ledger. The PCP theorem has been proposed as a potential method for improving the scalability of these ledgers, by making validation that a particular claim is valid answerable in a constant number of questions.

An interesting way to view the problem is that certain 'local' properties of mathematical objects can be encoded in such a way that these local properties become global. Here is an analogy from Dinur's talk on the subject. Imagine if we have a piece of toast, and, by sampling three points at random on the piece of toast, we want to determine whether the toast has jam on it. The encoding process as above can be compared to taking a knife, and smothering it all over the piece of toast, not looking at the bread. If there is any jam on the bread, then the knife will smother the jam all over the piece of toast, and after this, if we have any local jam on the piece of toast, it will now become a global property of the bread, and this can be very easily obtained by sampling three points on the piece of toast.

Example. *Three colorability of graphs is a very local property. In particular, a graph can appear to be three colorable, but fail to be three colorable at a single position. If someone said to us they had a three coloring of a graph, and we could only ask them the color of individual vertices, then it would take $|V|$ questions to determine whether the person actually had a three coloring. If you had access to the entire three coloring, it would only take you $O(|V| + |E|)$ steps to check whether a given coloring was actually a three coloring, so the problem of checking a three coloring is computationally feasible, and the PCP theorem guarantees that we can select three questions at random about the prover's proof of the three colorability of the graph, and to a high degree of accuracy, determine whether the prover's proof given is false, or whether the proof is correct. Note that "Do you have a three coloring", is NOT a good question to ask in this context, because we have no reason to trust the prover, and if he said "Yes" we have no information about his proof. Thus there is a way of encoding the three colorability of a graph in such a way that the local property of failing to be three colorable becomes global.*

Example. *The mathematical proof of a mathematical theorem is a very local property. Proofs can seem correct, but have a single 'bug' at a single location which invalidates the entire proof. The PCP theorem says that there is a new way of encoding these proofs such that bugs are globally expressed, and become easy to find.*

We shall find that Fourier-like techniques are very useful for transforming local properties of mathematical objects to global properties. An example of why this is true is that the L^1 norm of the Fourier transform of a function (which can be viewed as a particular encoding of a function) controls the L^∞ norm of the original function. The L^∞ norm is a very local property of the original function, whereas the L^1 norm of the Fourier transform is a global property, and can be estimated accuracy with a few random samples of the function.

1 Examples of PCPs

Here is some formal information which will be useful to reason about PCPs. Once we have predetermined a series of yes/no questions the verifier will pick from to determine if the theorem is correct, then we can index each question by an integer i, and then a prover can be identified with a finite series of zeroes and ones $x \in \{0,1\}^n$, such that $x_i = 1$ if the prover responds yes to question i, and $x_i = 0$ if the prover responds no to question i. If we ask three questions i, j, and k, we observe the bits x_i, x_j, and x_k, and must decide whether we are convinced by these three bits.

Example. *Consider determining whether two graphs G_0 and G_1 with n indexed nodes, are not isomorphic to one another. While it is easy to provide a proof that two graphs are isomorphic (just give me an isomorphism, and I can check that it is injective, bijective, and an isomorphism in linear time in the size of the graphs), it is not easy to show that two graphs are not isomorphic. In fact, we don't actually know whether the non isomorphism problem is feasibly checkable (though a proof that it isn't would imply $\mathbf{P} \neq \mathbf{NP}$). However, we can stll obtain an interactive proof of this result, which only asks a single question. Our questions will be of the following variety. Given a graph H, we ask*

"If H is isomorphic to G_0, output 0. If H is isomorphic to G_2, output 0. Otherwise, output an arbitrary result"

(Note that if the graphs are isomorphic, then this question isn't even well formed, and thus cannot be answered correctly). To form our question, we fix an index $i \in \{0,1\}$, and a permutation $\nu \in S_n$, chosen uniformly at random. Given the graph G_i, we form the graph $\nu(G_i)$ by permuting the indices of the nodes of G_i. Now we ask the question above for $H = \nu(G_i)$. If the answer is equal to i, we are convinced of the statement, and otherwise, we remain unconvinced. If the graphs are not isomorphic, and the prover answers correctly, we are convinced. Thus we have perfect completeness. Now suppose the graphs are isomorphic to one another. Then the random graph $\nu(G_i)$ we obtain is independent of the index i, and so for any answer the prover gives, there is a 50% chance of being caught out, because for each graph H we could pick to ask about, we expect either answer with a 50% probability. Thus we have imperfect soundness.

Thus encoding a graph as a partition of all graphs which are isomorphic and nonisomorphic to the graph makes the isomorphism property into a global property. This can be seen as a technique of 'smearing' a bunch of checks into a single check. Picking a graph and asking whether it is isomorphic to the original checks a bunch of edges all at once, and this enables us to make local properties global. One problem with this, however, is that the encoding increases the size of the encoding from about n bits to about n^n. This is related to the amount of randomness in our problem; in order to determine the probability distribution of our answers, we required a single 'coin flip' to determine our index i, but $\lg n!$ coin flips to determine $\nu \in S_n$. Thus we require $\lg n! + 1 = O(n \lg n)$ flips, which gives us the $2^{O(n \lg n)} = n^{O(n)}$ proof encoding. An additional fact provided by the

PCP theorem is that the random proofs for computationally feasible problems are that we only require $O(\lg n)$ coin flips of randomness for our problem, so that we can only ask at most $2^{O(\lg n)} = n^{O(1)}$ different questions, and so the size of our 'global' encodings differs from our original encodings by a polynomial amount. There are results analogous to the PCP theorem which show that any problem which can be checked in *exponential time* has an interactive proof asking K questions, but using a polynomial amount of randomness.

Example. *Here is an example of a* **local proof***, the Blum-Luby-Rosenfeld test, which have perfect completeness, but with local soundness, in the sense that the chance that we are tricked by the wrong solution depends on how 'close' the problem is to a real solution. This is the complete opposite of the checkable proofs we have considered previously, which require globally separated soundness. Suppose a prover has a function $f : \mathbf{F}_2^n \to \mathbf{F}_2$ they claim is linear. How many questions do we have to ask before are convinced it is linear. The proof is easy – we take two uniformly random $x, y \in \mathbf{F}_2^n$ (Note: $2 \log n = O(\log n)$ coin flips), and ask the value of $f(x)$, $f(y)$, and $f(x + y)$. We are convinced the function is linear if $f(x + y) = f(x) + f(y)$. If the function is linear, we always accept. On the other hand, for two functions, define the* **Hamming distance**

$$d(f, g) = \#\{x \in \mathbf{F}_2^n : f(x) \neq f(y)\}$$

Using Fourier-analytic methods, one can show that if $d(f, g) \geq \varepsilon$ for every linear function g, then f is rejected with probability $\geq \varepsilon$. Thus, though we can get tricked by functions which are close to linear functions, functions which disagree with all linear functions will not trick our tester. This reflects the fact that $d(f, g) = 1/2$ for any two linear functions f and g, so linear functions are very discrete in the Hamming metric. Suprisingly, if $d(f, g) = \varepsilon$, where g is some linear function, then using only the values of f, we can compute all the values of g correctly with probability $\geq 1 - 2\varepsilon$, even though f is different from g on a deterministic set of inputs. If we wish to calculate x, then we uniformly randomly pick y, and compute

$$f(x + y) - f(y)$$

$f(x + y) = g(x + y)$ with probability $1 - \varepsilon$, $f(y) = g(y)$ with probability $1 - \varepsilon$, and so by a union bound, with probability $\geq 1 - 2\varepsilon$, we find

$$f(x + y) - f(y) = g(x + y) - g(y) = g(x)$$

Thus linear functions are locally correctable, which enables us to move from local solutions to global solutions – thus, even though linearity is a local property, local correctability allows us to show that if solutions are accepted with high probability, they are close to correct, and in particular, this means there is a correct solution, and this is all that is important to be convinced by a proof.

Example. *Consider the problem of determining whether a series of quadratic equations over the field $\mathbf{F}_2$ is solvable, i.e. if*

$$x_1^2 + x_2 x_3 + x_4^2 = 1 \quad x_5^2 + x_2 x_6 + x_1^2 = 0$$

are solvable. This problem is **NP** *hard, which means I can check a solution fast, but if you can find a way to solve these equations in polynomial time, you get a million dollars. We will come up with a constant query checkable proof, which, by* **NP** *hardness, actually means we will have a constant query, probabilistically checkable proof for all feasibly checkable problems – this is the first step in proving the PCP theorem (we won't use a logarithmic amount of randomness though). Since $x_i^2 = x_i$, we may assume that all monomials in the equations have two terms. In this case, we can view the quadratic forms as obtained from a bilinear form on $\mathbf{F}_2^n$, and so we can find an $m \times n^2$ matrix A, and an m dimensional vector b, both with entries in $\mathbf{F}_2$, such that, if for $x \in \mathbf{F}_2^m$, we set $(x \otimes x)_{ij} = x_i x_j$, then we are solving $Ax = b$, where*

$$Ax = \sum A_{kij}(x \otimes x)_{ij} e_k = \sum A_{kij} x_i x_j e_k$$

We will provide an interactive proof of this result. In this case, the equations can fail locally (all but one equation can pass), so this is the first time we need to encode this problem in a nontrivial way that enables us to verify a result. We rely on discrete Fourier analysis type-results. We will ask questions of the following variety:

- *For some subset S of indices, what is $\sum_{i \in S} x_i$.*
- *For some subset J of pairs of indices, what is $\sum_{(i,j) \in J} x_i x_j$.*

Now how do we check that, when we ask our questions, we can really trust the answers? Note that for any subsets S and T,

$$\sum_{i \in S} x_i + \sum_{i \in T} x_i = \sum_{i \in S \Delta T} x_i$$

This is analogous to the linearity tester we just made, and indeed viewing the set of subsets of indices as a vector space over $\mathbf{F}_2$, this shows that the map $S \mapsto \sum_{i \in S} x_i$ should be linear. We start our checkable proof by picking S and T uniformly at random (Remark: uh oh, lots of randomness here!), and testing if the equation above holds, in three queries. By the last example, we know that this test can only pass with high probability if the sums we obtain are close in Hamming distance to an actual linear function, which we can interpret as a set of sums corresponding to some actual vector in $\mathbf{F}_2^n$, and we also know by local correctness that we can calculate the sums above corresponding to this x_i with high probability with three queries, so we can assume from the outset that the prover isn't lying! It takes three queries each to check that the two sums above correspond to actual values, so we may assume that the queries $\sum_{i \in S} x_i$ correspond to an actual $x \in \mathbf{F}_2^n$, and the queries $\sum_{(i,j) \in J} x_i x_j$ correspond to an actual vector $y \in \mathbf{F}_2^{\binom{n^2}{}}$. We don't yet know whether $x \otimes x = y$, though, so this is one way the prover could lie to us. However, we can check this, because if this was true, then we would find that for any two S, T,

$$\left(\sum_{i \in S} x_i \right) \left(\sum_{i \in T} x_j \right) = \sum_{(i,j) \in (S \times T)} x_i x_j$$

If we pick two random subsets S and T again, and check whether the sums agree with one another, then they might fail locally, but as with the proof of linearity, if the proof only fails in very particular locations, this means $x \otimes x$ and y are very close to one another, and we can use local correctibility to ensure $y = x \otimes x$. All that remains is to check that each of the quadratic equations was solvable by x. This is where all our work becomes powerful. It only takes a single query of the form

$$\sum_{(i,j) \in J} x_i x_j$$

to check whether a particular quadratic equation holds. If we only were able to ask about the values x_i, we would need to make at least n queries. We can't check all the equations, but again, there's a trick to globalize errors. We smear a bunch of equations together by randomly picking a subset.

2 Why are PCP testers useful?

From a practical, cryptologically perspective, it is easy to see why PCP testers are useful, because it means we can efficiently test whether malicious behaviour is going on in the transfer of information between parties. However, why do we care about it as pure mathematicians? The idea is the following, that we have referenced many times in this talk. The most important result in computing science, at least since Turing's incomputability result, is the following.

Theorem 3 (Cook-Levin Theorem)**.** *There exists problems, known as* **NP Complete Problems***, such that if we have any question that is easily checked, we can in polynomial time convert it to a question in the* **NP** *Complete problem. Thus, if an* **NP** *complete problem can be solved in polynomial time, then all problems in* **NP** *are easily solvable, and we conclude* $\mathbf{P} = \mathbf{NP}$*.*

The main example of an NP complete problem is, given a logical formula of the form

$$(x_1 \vee x_2 \vee x_4) \wedge (x_5 \vee x_1 \vee x_7) \wedge (x_1 \vee x_2 \vee x_4)$$

which is a bunch of ANDS of three variables OR'ed together, is it possible to assign truth values to these variables which will make this formula correct. We call this 3SAT. Because the problem is NP complete, it is unlikely we will ever find a solution to this problem that runs efficiently.

However, this isn't the end of the story, because we can change our problem, and rather than ask if the problem is *completely* solvable, ask what fraction of the clauses in our problem are solvable, making the problem an optimization problem under constraints. On the face of it, this problem is even more difficult than the original, because it asks us to obtain even more information than the original problem. However, we can cheat by asking ourself to come up with an approximation of the answer. We have an A **approximation algorithm** for this problem, for $A < 1$, if, whenver it is possible to satisfy B clauses, we can find an assignment efficiently that satisfies $\geq AB$ of the clauses. For instance, if

we assign each variable true or false uniformly at random, then the probability that each clause is satisfied is 7/8, so on average 7 out of 8 of clauses will be satisfied, and so we have a 7/8 approximation algorithm for 3SAT.

For problems that are NP hard, unless $P = NP$, we cannot find 1 approximation algorithms. However, the Cook-Levin theorem doesn't appear that we cannot find $1 - \varepsilon$ approximation algorithms for every ε. From the 70s onward, after Cook and Levin proved their result, people tried to adapt their proof to show that certain problems were hard enough that we could not approximate their solutions efficiently arbitrarily close. They attempted to adapt the proof of the Cook-Levin theorem to this problem, but they found a barrier which should now be familiar to us. The theorem encodes any problem as a 3SAT instance, but they found that if a theorem was false, the 3SAT instance that encoded it might still satisfy $1 - \varepsilon$ clauses. In other words, the 3SAT instance failed to be satisfiable locally. The PCP theorem was a major revolution to this approach, which allowed us to encode problems as 3SAT instances such that, if the problem is false, fail on at least δ clauses for some fixed, universal δ. This implies that we cannot have a $1 - \delta$ approximation algorithm to 3SAT unless $\mathbf{P} = \mathbf{NP}$, because an encoded solution is true if and only if it is true on at least $1 - \delta$ clauses, and so any approximate solution with at least $1 - \delta$ clauses guarantees the 3SAT instance is completely satisfiable. Thus we have a distinct barrier of approximability for 3SAT. The first version of the PCP theorem wasn't able to obtain what δ was, only that it existed. Using Håstad's improved version of the PCP theorem only making 3 queries, we can conclude that for any ε, if we had a $7/8 + \varepsilon$ approximation algorithm of the result, then $\mathbf{P} = \mathbf{NP}$. We have a 7/8 approximation algorithm for 3SAT, so the approximation problem for 3SAT is essentially solved. Almost all inapproximability results proved in the past two decades have reduced their problem to this result.