Report on the PCP Theorem

Omar Muhammad

November 18, 2025

Contents

Part I
Foundations and Statement of the Theorem

1 Fundamental Concepts in Computational Complexity

1.1 The Turing Machine

At its core, computation deals with the manipulation of strings. An **alphabet** is a finite, non-empty set of symbols, which for our purposes will almost always be the binary alphabet $\Sigma = \{0, 1\}$. A **string** is a finite sequence of symbols from an alphabet. The set of all possible finite-length strings over an alphabet Σ is denoted by Σ^*.

A **language** is any subset of Σ^*. A **decision problem** is the task of determining whether a given input string x is a member of a particular language L. An algorithm is said to *decide* a language L if, given any input string $x \in \Sigma^*$, it correctly determines whether or not $x \in L$.

1.1.1 Turing Machine Definition

The standard mathematical model for a computer is the Turing Machine. It consists of tapes, heads that read and write symbols on the tapes, and a finite set of internal states that dictate its actions. We use the multi-tape model as it more closely resembles an actual algorithm's use of memory.

Definition 1.1 (Turing Machine)**.** *A k-tape deterministic Turing Machine (TM) is formally described by a tuple $M = (\Gamma, Q, \delta)$, where:*

- *Γ is a finite set of symbols called the tape alphabet. We assume Γ contains a blank symbol $\sqcup$, a start symbol $\triangleright$, and the binary symbols $0, 1$.*
- *Q is a finite set of states, which includes a designated start state q_{start} and a halt state q_{halt}.*
- *δ is the transition function, with the signature:*

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$$

The machine's computation is performed on k tapes, which are one-way infinite sequences of cells. The first tape is the **read-only input tape**, while the other $k - 1$ tapes are **read/write work tapes**. The last of these, tape k, is also designated as the **output tape**.

The **start configuration** of a TM M on an input string x is defined as follows:

- The input tape contains the start symbol $\triangleright$, followed by the string x, followed by an infinite sequence of blank symbols $\sqcup$.
- All work tapes contain the start symbol $\triangleright$ in their first cell, followed by infinite blank symbols.
- All k tape heads start at the leftmost cell of their respective tapes.
- The machine is in the designated start state, q_{start}.

At each step of the computation, if the machine is in state $q \in Q$ and the heads are reading symbols $(\sigma_1, \sigma_2, \ldots, \sigma_k) \in \Gamma^k$ from the k tapes, and the transition function dictates $\delta(q, (\sigma_1, \ldots, \sigma_k)) = (q', (\sigma'_2, \ldots, \sigma'_k), z)$, then in the next step:

1. The machine's state changes to q'.
2. For each $i \in \{2, \ldots, k\}$, the symbol σ'_i is written on tape i. The input tape is not written to.
3. The k tape heads move according to the vector $z \in \{L, S, R\}^k$ (Left, Stay, Right).

The machine halts if it enters the state q_{halt}.

1.1.2 Nondeterministic Turing Machines

We now define a variant of the Turing Machine that incorporates the concept of "nondeterministic choice."

Definition 1.2 (Nondeterministic Turing Machine)**.** *A Nondeterministic Turing Machine (NDTM) is a modification of the deterministic model. The formal differences are:*

1. *The machine has **two** transition functions, denoted δ_0 and δ_1.*
2. *Its set of states Q includes a special state q_{accept}, which is distinct from the regular halting state q_{halt}.*

At each step of its computation, an NDTM makes an arbitrary choice between applying δ_0 or δ_1. This sequence of choices generates a tree of possible computation paths. The machine's output is defined based on the existence of at least one successful path.

- An NDTM M **accepts** an input x if there *exists* at least one sequence of nondeterministic choices that leads the machine to the state q_{accept}.
- An NDTM M **rejects** an input x if *every* possible sequence of choices leads the machine to the state q_{halt}.

1.1.3 Running Time and Efficiency

The efficiency of a computation is measured by the number of elementary steps a Turing Machine takes to halt, expressed as a function of its input length.

Definition 1.3 (Running Time of a Deterministic TM)**.** *Let M be a deterministic Turing Machine that halts on all inputs. The **running time** of M is the function $T : \mathbb{N} \to \mathbb{N}$, where $T(n)$ is the **maximum** number of steps that M executes on any input of length n.*

Definition 1.4 (Computing a Function in $T(n)$-time)**.** *Let $f : \{0,1\}^* \to \{0,1\}^*$ and $T : \mathbb{N} \to \mathbb{N}$ be some functions. A TM M **computes** f **in** $T(n)$**-time** if for every input string $x \in \{0,1\}^*$, M, when initialized to the start configuration on input x, halts within $T(|x|)$ steps with the string $f(x)$ written on its output tape.*

For a *nondeterministic* TM, the definition of running time is extended to consider all possible computation paths.

Definition 1.5 (Running Time of an NDTM)**.** *An NDTM M runs in $T(n)$ time if for every input x of length n, **every** possible computation path of M halts (in either q_{accept} or q_{halt}) within $T(|x|)$ steps.*

These formal definitions of running time allow us to group problems into classes based on their computational resource requirements, which is the subject of the next section.

1.2 P and NP

Definition 1.6. *We say a Turing machine M* decides *a language $L \subseteq \{0,1\}^*$ if it computes the function $f_L : \{0,1\}^* \to \{0,1\}$, where $f_L(x) = 1 \iff x \in L$.*

1.2.1 The Class P

Definition 1.7 (DTIME)**.** *DTIME($T(n)$) be the set of all languages decidable by a TM in $O(T(n))$ time.*

Definition 1.8 (P)**.** *The complexity class* ***P*** *is the set of all languages that can be decided by a deterministic Turing Machine in polynomial time. Formally:*

$$\boldsymbol{P} = \bigcup_{c \geq 1} DTIME(n^c)$$

1.2.2 The Class NP

Historically, the class NP was defined based on the Nondeterministic Turing Machine model.

Definition 1.9 (NTIME)**.** *For a function $T : \mathbb{N} \to \mathbb{N}$, the class $NTIME(T(n))$ is the set of all languages L that can be decided by a Nondeterministic Turing Machine running in time $O(T(n))$.*

Definition 1.10 (NP)**.** *The complexity class* ***NP*** *is the set of all languages that can be decided by a Nondeterministic Turing Machine in polynomial time. Formally:*

$$\boldsymbol{NP} = \bigcup_{c \geq 1} NTIME(n^c)$$

1.2.3 The P versus NP Question

The relationship between deterministic and nondeterministic computation is foundational to complexity theory.

Lemma 1.11. $\boldsymbol{P \subseteq NP}$.

Proof. A deterministic TM is a special case of an NDTM where the two transition functions are identical ($\delta_0 = \delta_1$). This leads to a clear containment. □

This containment brings us to the most significant open question in computer science: does P equal NP? This asks whether the power of nondeterministic "guessing" provides any true computational speed-up over deterministic computation for solving problems. It is widely conjectured that $\mathbf{P} \neq \mathbf{NP}$.

1.3 NP as a Verifiable Proof System

While the NDTM definition is formal, a more intuitive and practically useful definition of NP is based on the concept of efficient verification. This view introduces the idea of a "proof" or "certificate" for a YES answer.

Definition 1.12 (The Class NP - Verifier Version)**.** *A language $L \subseteq \{0,1\}^*$ is in* ***NP*** *if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a deterministic, polynomial-time Turing Machine M, called the* ***verifier****, such that for every string $x \in \{0,1\}^*$:*

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)} \text{ such that } M(x,u) = 1$$

The string u is called a ***certificate*** *or* ***witness*** *for the membership of x in L.*

Note on Turing Machine Inputs While the verifier M is written as a function of two inputs, (x, u), our formal Turing Machine model takes a single string on its input tape. This is handled by a standard encoding convention. The pair (x, u) is first represented as a single string by concatenating them with a special separator symbol, such as $x\#u$. This new string, which is over the alphabet $\{0, 1, \#\}$, is then converted into a purely binary string using a mapping like $0 \mapsto 00$, $1 \mapsto 11$, and $\# \mapsto 01$. This allows us to rigorously treat $M(x, u)$ as a TM operating on a single input without any loss of generality.

Theorem 1.13. *The NDTM-based and verifier-based definitions of NP are equivalent.*

Proof. let us denote the class of languages decidable by a polynomial-time NDTM as **NP** and the class of languages with a polynomial-time verifier as **NP'**. We will show that **NP** = **NP'**.

Part 1: NP' ⊆ NP Let L be an arbitrary language in **NP'**. Let its corresponding polynomial be $p : \mathbb{N} \to \mathbb{N}$ and deterministic, polynomial-time Turing Machine be M (the verifier). We will construct a polynomial-time NDTM, let's call it N, that decides L. The machine N operates as follows on an input x:

1. Using its nondeterministic choices, N writes a string u of length $p(|x|)$ onto one of its work tapes. This takes $p(|x|)$ steps. Each possible string of this length corresponds to one computation path.
2. After generating u, N proceeds deterministically. It simulates the verifier M on the combined input (x, u).
3. If the simulation of M halts and outputs 1, then N transitions to its own accept state, q_{accept}. Otherwise, N halts without accepting.

Correctness: If $x \in L$, then by the definition of **NP'**, there must exist at least one certificate u_0 such that $M(x, u_0) = 1$. The NDTM N has a computation path corresponding to guessing this specific certificate u_0. On this path, the simulation of M will accept, causing N to accept. Therefore, N accepts x. Conversely, if $x \notin L$, no certificate u exists for which $M(x, u) = 1$. Consequently, no matter which string u is generated by N, the simulation of M will reject, and no computation path of N will lead to q_{accept}.

Running Time: The first step takes $p(|x|)$ time. The second step involves simulating a polynomial-time machine M on an input of size $|x| + p(|x|)$, which is also polynomial in $|x|$. The total time is the sum of two polynomials, which is itself a polynomial. Thus, N is a polynomial-time NDTM.

Since we have constructed a polynomial-time NDTM for any language $L \in$ **NP'**, we have shown that $L \in$ **NP**. Therefore, **NP' ⊆ NP**.

Part 2: NP ⊆ NP' Now, let L be an arbitrary language in **NP**. By definition, there exists a polynomial-time NDTM, let's call it N, that decides L. Let the running time of N be bounded by a polynomial $p(|x|)$.

The certificate u for an input x will be a string of length at most $p(|x|)$, where each bit corresponds to a nondeterministic choice made by N (e.g., '0' for δ_0, '1' for δ_1).

The deterministic verifier M is constructed as follows. On input (x, u):

1. M simulates the NDTM N on input x.
2. At each step i of the simulation where N would make a nondeterministic choice, M instead reads the i-th bit of the certificate string u. It uses this bit to deterministically choose which transition function (δ_0 or δ_1) to apply.
3. M runs this simulation for at most $p(|x|)$ steps.
4. If the simulation of N reaches the state q_{accept}, then M halts and outputs 1. If the simulation halts without accepting, or if the certificate u is exhausted before the simulation completes, M halts and outputs 0.

Correctness: If $x \in L$, then by definition of **NP**, there must exist an accepting computation path for N. The sequence of choices for that path is a valid certificate, let's call it u_0. When M is given (x, u_0), its simulation will follow this exact path and accept. Conversely, if $x \notin L$, no accepting path exists for N, so no certificate u can guide the simulation of M to an accepting state.

Running Time: The certificate u has polynomial length $p(|x|)$. The verifier M simulates each step of N in a fixed number of its own steps. Since N runs in polynomial time, the simulation also runs in polynomial time. Thus, M is a polynomial-time verifier.

Since we have constructed a valid verifier for any language $L \in$ **NP**, we have shown that $L \in$ **NP'**. Therefore, **NP** $\subseteq$ **NP'**.

Conclusion: Since **NP'** $\subseteq$ **NP** and **NP** $\subseteq$ **NP'**, the two classes are identical. □

1.3.1 Examples of Problems in P and NP

The verifier-based definition makes it particularly easy to classify many natural problems.

- **Problems in P:**
  - **Connectivity:** Given a graph G and two vertices s, t, is there a path between them? This can be solved in polynomial time using algorithms like Breadth-First Search (BFS).
  - **Primality Testing:** Given an integer N, is it a prime number? Famously shown to be in P by the AKS primality test.
- **Problems in NP (and not known to be in P):**
  - **SAT (Boolean Satisfiability):** Given a Boolean formula, is there an assignment of TRUE/FALSE values to its variables that makes the entire formula TRUE? The *certificate* is a satisfying assignment.
  - **INDSET (Independent Set):** Given a graph G and an integer k, does G contain an independent set of size at least k? The *certificate* is the set of k vertices itself.
  - **TSP (Traveling Salesperson Problem):** Given a list of cities and distances, is there a tour visiting every city with a total length no more than k? The *certificate* is the sequence of cities in such a tour.

1.4 NP-Completeness and Reductions

Within the class NP, certain problems are considered the "hardest" in the sense that a polynomial-time algorithm for any one of them would imply a polynomial-time algorithm for every problem in NP. This concept is formalized through polynomial-time reductions.

1.4.1 Reductions

A reduction is a method for solving one problem using an algorithm for another. If we can efficiently transform instances of problem A into instances of problem B in a way that preserves the answer, then we can say that B is at least as hard as A.

Definition 1.14 (Polynomial-Time Karp Reduction)**.** *A language $A \subseteq \{0,1\}^*$ is **polynomial-time Karp reducible** to a language $B \subseteq \{0,1\}^*$, denoted $A \leq_p B$, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$:*

$$x \in A \iff f(x) \in B$$

1.4.2 NP-Hardness and NP-Completeness

Using this definition of a reduction, we can formally define the hardest problems in NP.

Definition 1.15 (NP-Hard and NP-Complete)**.** *A language B is **NP-hard** if $A \leq_p B$ for every language $A \in$ **NP**. A language B is **NP-complete** if it is NP-hard and $B \in$ **NP**.*

NP-complete problems are thus the problems that are both in NP and are at least as hard as any other problem in NP. Their significance to the P versus NP question is profound, as captured by the following theorem.

Theorem 1.16. *Let A be an NP-complete language. Then $\boldsymbol{P = NP}$ if and only if $A \in \boldsymbol{P}$.*

Proof. If $\mathbf{P} = \mathbf{NP}$, then since $A \in \mathbf{NP}$, it must be that $A \in \mathbf{P}$.

Conversely, suppose $A \in \mathbf{P}$. This means there is a deterministic TM that decides A in polynomial time. Since A is NP-hard, every language $L \in \mathbf{NP}$ is reducible to A via some polynomial-time computable function f.

This gives us a polynomial-time algorithm to decide any language $L \in \mathbf{NP}$:

1. On an input x, first compute the string $y = f(x)$. Since f is polynomial-time computable, this step runs in polynomial time in $|x|$, and the length of the output string y is also at most a polynomial in $|x|$.
2. Next, run the polynomial-time decider for A on the input y. Since the runtime of this decider is polynomial in $|y|$, and $|y|$ is polynomial in $|x|$, this step also runs in polynomial time in $|x|$.

The composition of two polynomial-time computations is polynomial, so we have a polynomial-time algorithm for L. Since this holds for all $L \in \mathbf{NP}$, we have shown that $\mathbf{NP} \subseteq \mathbf{P}$. Combined with the fact that $\mathbf{P} \subseteq \mathbf{NP}$, this implies $\mathbf{P} = \mathbf{NP}$. □

1.4.3 The Cook-Levin Theorem

The question then arises: do NP-complete problems even exist? The landmark Cook-Levin theorem was the first to establish that they do, by identifying a natural combinatorial problem that is NP-complete. To state it, we first need to define Boolean formulas.

A **Boolean formula** consists of Boolean variables $(u_1, u_2, \dots)$ and logical operators AND $(\wedge)$, OR $(\vee)$, and NOT $(\neg)$. A formula is in **Conjunctive Normal Form (CNF)** if it is an AND of clauses, where each clause is an OR of literals (a variable or its negation). A **k-CNF** formula is a CNF formula where each clause has at most k literals.

A Boolean formula is said to be **satisfiable** if there exists an assignment to the variables such that the formula is TRUE.

Definition 1.17 (The SAT and 3SAT Problems)**.**

- ***SAT*** *is the language of all satisfiable CNF formulas.*
- ***3SAT*** *is the language of all satisfiable 3-CNF formulas.*

Theorem 1.18 (Cook-Levin Theorem)**.** *The languages SAT and 3SAT are both NP-complete.*

Proof. The proof proceeds in three main stages. First, we establish that SAT is in NP. Second, we prove that SAT is NP-hard by reducing any NP problem to it. Finally, we show that 3SAT is NP-hard by reducing SAT to 3SAT.

1. SAT is in NP To show SAT is in NP, we must provide a polynomial-time verifier.

- **Certificate:** A proposed satisfying assignment for the variables of the input formula ϕ. This certificate's length is equal to the number of variables.
- **Verifier:** A deterministic TM that substitutes the certificate's assignment into ϕ and evaluates each clause. If all clauses are true, it accepts; otherwise, it rejects. This process takes polynomial time in the size of ϕ.

2. SAT is NP-hard To prove NP-hardness, we show that any language $L \in \mathbf{NP}$ can be reduced to SAT. Let M be the polynomial-time NDTM that decides L. For any input x, we construct a polynomial-sized CNF formula ϕ_x that is satisfiable if and only if M accepts x.

The formula ϕ_x simulates the computation of M on x. We represent the entire computation history of M as a *tableau*—a grid where rows represent time steps and columns represent tape cells. The key insight is the **locality of computation**: the state of a cell at time $t+1$ depends only on its own state and that of its immediate neighbors at time t.

We define Boolean variables to represent the state of this tableau at each time step (e.g., 'STATE(t, q)', 'HEAD(t, i)', 'TAPE(t, i, s)'). The formula ϕ_x is a conjunction of four sets of clauses that enforce a valid, accepting computation:

1. ϕ_{start}**:** Clauses fixing the tableau's configuration at time $t = 0$ to match the start configuration of M on input x.

2. ϕ_{accept}**:** A clause asserting that the machine is in an accepting state, q_{accept}, at the final time step.

3. ϕ_{move}**:** Clauses ensuring that each configuration legally follows from the previous one according to M's transition function. This is possible due to the locality of computation.

4. ϕ_{unique}**:** Clauses enforcing consistency, ensuring the machine is in exactly one state and has its head in one position at any given time.

A satisfying assignment for $\phi_x = \phi_{start} \wedge \phi_{accept} \wedge \phi_{move} \wedge \phi_{unique}$ directly corresponds to a valid, accepting computation path of M. The construction of ϕ_x takes polynomial time, completing the reduction.

3. SAT $\leq_p$ 3SAT Since 3SAT is a special case of SAT, it is also in NP. To show it is NP-hard, we reduce SAT to it. We provide a polynomial-time transformation that converts any CNF formula ϕ into an equisatisfiable 3-CNF formula ϕ'. Any clause in ϕ with three or fewer literals is kept as is. For any clause $C = (l_1 \vee l_2 \vee \cdots \vee l_k)$ with $k > 3$, we replace it with a set of smaller clauses by introducing new "dummy" variables. For example, a clause $(l_1 \vee l_2 \vee l_3 \vee l_4)$ is replaced by the conjunction of two clauses:

$$(l_1 \vee l_2 \vee z) \wedge (\neg z \vee l_3 \vee l_4)$$

where z is a new variable. This new expression is satisfiable if and only if the original clause was. This process is repeated until all clauses have at most 3 literals. The resulting formula ϕ' is in 3-CNF and is only polynomially larger than ϕ. By transitivity of reductions, since SAT is NP-hard, 3SAT must also be NP-hard, and is therefore NP-complete. □

2 The PCP Theorem: Statement

2.1 The Probabilistic Verifier View

The traditional definition of NP relies on a deterministic verifier that must read the entire certificate to confirm its validity. The PCP theorem introduces a new, powerful type of verifier that is both probabilistic and remarkably efficient in its access to the proof.

This verifier is given random access to a proof string $\pi \in \{0,1\}^*$. Instead of reading the whole proof, it uses a string of random bits to select a small number of positions in π, queries those positions, and decides whether to accept or reject based only on the bits it reads.

Definition 2.1 (Probabilistically Checkable Proof (PCP) Verifier)**.** *For functions $r, q : \mathbb{N} \to \mathbb{N}$, an $(r(n), q(n))$-**verifier** for a language L is a polynomial-time probabilistic Turing Machine V with the following properties for any input x of length n:*

- ***Efficiency:*** *V uses at most $r(n)$ random bits to select and make at most $q(n)$ non-adaptive queries (i.e. queries only depend on the input and random bits) to a proof string $\pi \in \{0,1\}^*$. Based on the results, it deterministically outputs either 1 (accept) or 0 (reject). We denote the output, which is a random variable, by $V^\pi(x)$.*

- ***Completeness (Perfect):*** *If $x \in L$, then there exists a proof string $\pi \in \{0,1\}^*$ such that*

the verifier always accepts:

$$\Pr[V^\pi(x) = 1] = 1$$

- ***Soundness:*** *If $x \notin L$, then for **any** purported proof string $\pi \in \{0,1\}^*$, the verifier accepts with probability at most $1/2$:*

$$\Pr[V^\pi(x) = 1] \leq \frac{1}{2}$$

This leads to the definition of a new complexity class based on these resources.

Definition 2.2 (The Class PCP)**.** *A language L is in the class $\boldsymbol{PCP}(r(n), q(n))$ if it has an $(O(r(n)), O(q(n)))$-verifier.*

With this formalism, we can now state the celebrated PCP Theorem, which provides a powerful and surprising new characterization of the class NP.

Theorem 2.3 (The PCP Theorem)**.**

$$\boldsymbol{NP} = \boldsymbol{PCP}(\log n, 1)$$

Lemma 2.4 (Containment of PCP in NTIME)**.** *For any time-constructible functions $r(n)$ and $q(n)$,*

$$\mathrm{PCP}(r(n), q(n)) \subseteq \mathrm{NTIME}\big(2^{O(r(n))} \cdot (q(n) + \mathrm{poly}(n))\big).$$

Proof. Let L be a language in $\mathbf{PCP}(r(n), q(n))$, and let V be its corresponding PCP verifier. We can construct an NDTM, let's call it N, that decides L.

The verifier V uses at most $c \cdot r(n)$ random bits and makes at most $d \cdot q(n)$ queries for some constants c, d. The total number of locations in the proof π that could possibly be queried by V is at most $d \cdot q(n) \cdot 2^{c \cdot r(n)}$. Any other bits in π are irrelevant. Therefore, the effective length of the proof is bounded by $2^{O(r(n))} q(n)$.

The NDTM N operates as follows on input x with length n:

1. N nondeterministically guesses the entire effective proof string π, which has a length of at most $2^{O(r(n))} q(n)$.
2. N then deterministically simulates the verifier V on (x, π). It does this by iterating through all $2^{c \cdot r(n)}$ possible random strings. For each random string, it simulates the queries and the decision of V.
3. If the simulation of V accepts for *every* possible random string, then N transitions to its own accept state, q_{accept}. Otherwise, it rejects.

Let's analyze the time complexity of the verification phase.

- The number of iterations is $2^{c \cdot r(n)} = 2^{O(r(n))}$.
- In each iteration, N simulates V. Since V is a polynomial-time machine by definition, one simulation takes $O(n^k)$ time for some fixed constant k (the runtime of V also depends on $q(n)$, but this is absorbed into the polynomial bound).
- Therefore, the total time for the verification phase is the product of these two factors: $O(n^k \cdot 2^{O(r(n))})$.

The total running time of the NDTM N is the sum of the times for the guessing and verification phases:

$$T_{total}(n) = O(q(n) \cdot 2^{O(r(n))}) + O(n^k \cdot 2^{O(r(n))})$$

$$T_{total}(n) = O((q(n) + n^k) \cdot 2^{O(r(n))})$$

□

This general lemma directly implies the "easy direction" of the PCP Theorem.

Theorem 2.5 (The "Easy Direction" of the PCP Theorem)**.** ***PCP***$(\log n, 1) \subseteq$ ***NP***.

Proof. This is a direct corollary of the previous lemma. We substitute $r(n) = \log n$ and $q(n) = 1$ into the total time complexity bound we derived:

$$T_{total}(n) = O((1 + n^k) \cdot 2^{O(\log n)})$$

$$T_{total}(n) = O(n^k \cdot n^{O(1)}) = O(n^{k+c}) = \text{poly}(n)$$

Since the NDTM runs in polynomial time, the language $L \in PCP(\log n, 1)$ is in $\mathbf{NTIME}(n^{O(1)})$, which is, by definition, the class **NP**. Therefore, $\mathbf{PCP}(\log n, 1) \subseteq \mathbf{NP}$. □

Theorem 2.6 (Soundness Amplification by Repetition)**.** *Let L be a language in* ***PCP***$(r(n), q(n))$ *with soundness error $s < 1$. Then for any integer constant $k \geq 1$, the language L is also in* ***PCP***$(r(n), q(n))$ *with soundness error s^k.*

Proof. Let V be the original $(r(n), q(n))$-verifier for L with soundness s. We construct a new verifier, V', that achieves the improved soundness.

Construction of the New Verifier V' On input x and with access to a proof π, the verifier V' performs the following procedure:

1. Repeat the following for k independent trials:
   (a) Generate a new, independent random string of length $r(n)$.
   (b) Use this random string to simulate one run of the original verifier V on input (x, π).
2. The new verifier V' accepts if and only if **all** k trials of V accepted. If even one trial rejects, V' rejects.

We now analyze the properties of this new verifier V'.

Completeness If $x \in L$, there exists a correct proof π such that $\Pr[V^\pi(x) = 1] = 1$. Since each of the k trials of V' is just a run of V, each trial will accept with probability 1. The probability that all k trials accept is therefore $1^k = 1$. Thus, the completeness condition is maintained.

Soundness If $x \notin L$, then for any purported proof string π, the original verifier V accepts with probability at most s. Since the k trials performed by V' are independent (because they use fresh random strings each time), the probability that all of them accept is the product of their individual probabilities:

$$\Pr[V' \text{ accepts}] = \Pr[\text{Trial 1 accepts } \wedge \cdots \wedge \text{ Trial k accepts}]$$

$$\Pr[V' \text{ accepts}] = \prod_{i=1}^{k} \Pr[\text{Trial i accepts}]$$

$$\Pr[V' \text{ accepts}] \leq s \cdot s \cdot \cdots \cdot s = s^k$$

The soundness of the new verifier V' is therefore at most s^k.

Resource Usage

- **Randomness:** The verifier V' uses k independent random strings, each of length $r(n)$. The total number of random bits is $k \cdot r(n)$.
- **Queries:** The verifier V' makes $q(n)$ queries in each of its k trials. The total number of queries is at most $k \cdot q(n)$.

Since k is a constant, the new resource bounds are $O(r(n))$ and $O(q(n))$, respectively. Thus, we have constructed a valid PCP verifier for L with the claimed parameters and improved soundness. □

2.2 The Hardness of Approximation View

While the PCP Theorem provides a new characterization of NP in terms of proof checking, its most significant impact has been in the field of computational hardness of approximation. It implies that for many NP-hard optimization problems, finding even a "good enough" approximate solution is just as difficult as finding the exact, optimal solution.

2.2.1 Approximation Algorithms and MAX-3SAT

Many NP-complete decision problems have natural optimization counterparts. For 3SAT, the corresponding optimization problem is MAX-3SAT.

Definition 2.7 (MAX-3SAT)**.** *Given a 3-CNF formula ϕ, the **MAX-3SAT** problem is to find a variable assignment that maximizes the number of satisfied clauses.*

For a given formula ϕ, we define $val(\phi)$ as the maximum fraction of clauses that can be satisfied by any single assignment. If ϕ is satisfiable, then $val(\phi) = 1$. An approximation algorithm seeks to find an assignment that is provably close to this optimal value.

Definition 2.8 (Approximation Algorithm)**.** *For a constant $\rho \leq 1$, a polynomial-time algorithm A is a **ρ-approximation algorithm** for MAX-3SAT if for every 3-CNF formula ϕ, $A(\phi)$ outputs an assignment that satisfies at least $\rho \cdot val(\phi)$ fraction of the clauses.*

An important baseline result is that a simple, deterministic greedy algorithm can achieve a 1/2-approximation for MAX-3SAT. This provides a worst-case guarantee that fits our definition of an approximation algorithm.

Theorem 2.9 (Greedy 1/2-Approximation for MAX-3SAT)**.** *There exists a polynomial-time, deterministic 1/2-approximation algorithm for MAX-3SAT.*

Proof. Let the given 3-CNF formula be ϕ with variables $x_1, x_2, \ldots, x_n$ and m clauses. The algorithm iterates through the variables one by one, making a greedy choice for each.

Algorithm Description Initialize an empty assignment A. For $i = 1, \ldots, n$:

1. Consider the variable x_i.
2. Evaluate the effect of setting x_i = TRUE versus x_i = FALSE on the currently unsatisfied clauses that contain x_i or $\neg x_i$.
3. Choose the assignment for x_i that satisfies the maximum number of such clauses.
4. Add this assignment to A and permanently remove all clauses that are now satisfied from the formula.

After iterating through all variables, the final assignment is A. This algorithm is clearly deterministic and runs in polynomial time.

Proof of Approximation Ratio Let A be the assignment produced by the greedy algorithm, and let S_A be the set of clauses it satisfies.

The algorithm processes variables in the order $x_1, x_2, \ldots, x_n$. At each step i, the algorithm makes a decision for x_i and satisfies a set of previously unsatisfied clauses, which we will call S_i. Since clauses are removed once satisfied, the sets S_i are disjoint, and the total number of clauses satisfied by the algorithm is $|S_A| = \sum_{i=1}^{n} |S_i|$.

At the beginning of step i, let C_i^+ be the set of currently unsatisfied clauses that contain the literal x_i, and let C_i^- be the set containing $\neg x_i$. The algorithm chooses the assignment for x_i that satisfies the larger of these two sets. Therefore, the number of clauses satisfied at step i is:

$$|S_i| = \max(|C_i^+|, |C_i^-|)$$

Using the fact that for any non-negative numbers a, b, we have $\max(a, b) \geq \frac{1}{2}(a + b)$, we get:

$$|S_i| \geq \frac{1}{2}\left(|C_i^+| + |C_i^-|\right)$$

Summing this inequality over all steps $i = 1, \ldots, n$:

$$|S_A| = \sum_{i=1}^{n} |S_i| \geq \sum_{i=1}^{n} \frac{1}{2}\left(|C_i^+| + |C_i^-|\right) = \frac{1}{2}\sum_{i=1}^{n}\left(|C_i^+| + |C_i^-|\right)$$

The term $|C_i^+| + |C_i^-|$ counts the number of unsatisfied clauses at the start of step i that contain either x_i or $\neg x_i$. Every clause C in the original formula contains some set of variables. Let x_k be the variable in C with the lowest index k. At step k, C that has not yet been satisfied, it will be a member of either C_k^+ or C_k^-. Once step k is complete, C will either be satisfied and removed, or it will remain, but it will never be counted again in a future step $j > k$ because its lowest-indexed variable has already been processed. Therefore, each of the m clauses is counted exactly once in this sum over all steps.

$$\sum_{i=1}^{n}\left(|C_i^+| + |C_i^-|\right) = m$$

Substituting this back into our inequality for $|S_A|$:

$$|S_A| \geq \frac{1}{2}m$$

Let $m^* = val(\phi) \cdot m$ be the number of clauses satisfied by an optimal assignment. By definition, $m^* \leq m$. Therefore:

$$|S_A| \geq \frac{1}{2}m \geq \frac{1}{2}m^*$$

This proves that the greedy algorithm provides a 1/2-approximation. □

The crucial question is whether we can get arbitrarily close to the optimal value. The PCP Theorem answers this with a resounding "no" (unless P=NP).

Theorem 2.10 (Hardness of Approximating MAX-3SAT)**.** *There exists a constant $\rho < 1$ such that if there is a polynomial-time ρ-approximation algorithm for MAX-3SAT, then $\boldsymbol{P = NP}$.*

The hardness result for MAX-3SAT is a direct consequence of a gap-producing reduction, which is a core part of the PCP theorem's hardness of approximation view.

Theorem 2.11 (Gap-Producing Reduction for 3SAT)**.** *There exists a constant $\rho < 1$ and a polynomial-time function f that maps strings to 3-CNF formulas, such that for any language $L \in \boldsymbol{NP}$ and any input string x:*

1. *If $x \in L \implies val(f(x)) = 1$.*

2. *If* $x \notin L \implies val(f(x)) < \rho$.

Proof. The proof of this theorem is equivalent to the proof of the PCP Theorem itself and will be the subject of the remainder of this report. For now, we assume its correctness to prove the inapproximability of MAX-3SAT. □

We now prove Theorem 2.10.

Proof of Theorem 2.10. Assume that such a ρ-approximation algorithm, let's call it A, exists. We will use A to construct a polynomial-time algorithm that decides 3SAT, which is an NP-complete problem. By the properties of NP-completeness, this would imply that $\mathbf{P} = \mathbf{NP}$.

Let ϕ be any 3-CNF formula that is an input to the 3SAT decision problem. Our algorithm to decide 3SAT is as follows:

1. Use the polynomial-time, gap-producing reduction f from the previous theorem to transform ϕ into a new 3-CNF formula $\phi' = f(\phi)$.
2. Run the hypothetical ρ-approximation algorithm A on the formula ϕ'. Let the assignment returned by A be α.
3. Count the fraction of clauses in ϕ' that are satisfied by the assignment α. Let this fraction be s.
4. **Decision Rule:** If $s \geq \rho$, output YES (i.e., ϕ is satisfiable). Otherwise, output NO.

This entire algorithm runs in polynomial time. Now, we must prove its correctness by analyzing the two cases for the original formula ϕ.

Case 1: ϕ is satisfiable. By the completeness property of the reduction f, the new formula $\phi' = f(\phi)$ is also satisfiable, so $val(\phi') = 1$. Our approximation algorithm A, when run on ϕ', is guaranteed to return an assignment α that satisfies at least $\rho \cdot val(\phi')$ fraction of the clauses. Therefore, the fraction of satisfied clauses s will be at least $\rho \cdot 1 = \rho$. Our algorithm correctly outputs YES.

Case 2: ϕ is not satisfiable. By the soundness property of the reduction f, the new formula $\phi' = f(\phi)$ has $val(\phi') < \rho$. This means that no assignment can satisfy more than a fraction ρ of its clauses. The assignment α returned by algorithm A (or any other assignment) must therefore satisfy a fraction of clauses $s < \rho$. Our algorithm correctly outputs NO.

Since our polynomial-time algorithm correctly decides 3SAT, it follows that $3SAT \in \mathbf{P}$. Because 3SAT is NP-complete, this implies $\mathbf{P} = \mathbf{NP}$. □

2.3 The Satisfiability Gap Problem

We now formally prove that the probabilistic verifier view and the hardness of approximation view of the PCP Theorem are equivalent. This is best achieved by generalizing from 3SAT to the broader class of Constraint Satisfaction Problems (CSPs).

2.3.1 Constraint Satisfaction Problems (CSPs)

Definition 2.12 (Constraint Satisfaction Problem)**.** *A **q-CSP** instance ϕ is a collection of functions, called constraints, $\phi_1, \ldots, \phi_m$, from $\{0,1\}^n \to \{0,1\}$. Each function ϕ_i depends on at most q of its input locations (variables). That is, for each $i \in [m]$, there exist indices $j_1, \ldots, j_q \in [n]$ and a function $f : \{0,1\}^q \to \{0,1\}$ such that for every $u \in \{0,1\}^n$, $\phi_i(u) = f(u_{j_1}, \ldots, u_{j_q})$. We say an assignment $u \in \{0,1\}^n$ satisfies a constraint ϕ_i if $\phi_i(u) = 1$. The fraction of constraints satisfied by u is $\frac{1}{m}\sum_{i=1}^{m} \phi_i(u)$. The **value** of the instance, denoted $val(\phi)$, is the maximum fraction of constraints that can be satisfied by any single assignment. We say ϕ is satisfiable if $val(\phi) = 1$. The integer q is called the **arity** of the instance.*

2.3.2 The Gap Problem

Using this language, we can formally define the "gap" that is central to the hardness of approximation.

Definition 2.13 (Gap CSP). *For $q \in \mathbb{N}$ and $\rho \leq 1$, the **ρ-GAP-qCSP** is the problem of determining for a given q-CSP instance ϕ whether:*

1. *$val(\phi) = 1$ (in which case we say ϕ is a YES instance), or*
2. *$val(\phi) < \rho$ (in which case we say ϕ is a NO instance).*

*We say that ρ-GAP-qCSP is **NP-hard** if for every language $L \in \mathbf{NP}$, there is a polynomial-time function f mapping strings to q-CSP instances such that:*

- ***Completeness:*** *$x \in L \implies val(f(x)) = 1$.*
- ***Soundness:*** *$x \notin L \implies val(f(x)) < \rho$.*

2.3.3 Equivalence of the Two PCP Formulations

The two views of the PCP Theorem are both equivalent to the NP-hardness of a Gap-CSP problem.

Theorem 2.14 (Equivalence of Formulations). *The following statements are equivalent:*

1. ***NP*** $= \boldsymbol{PCP}(\log n, 1)$.
2. *There exist constants $q \in \mathbb{N}$ and $\rho < 1$ such that ρ-GAP-qCSP is NP-hard.*

Proof. We prove the equivalence by showing a construction for each direction.

Proof of (1 $\implies$ 2) Assume $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$. This means every language $L \in \mathbf{NP}$ has a PCP verifier V with $c \log n$ random bits and q queries for some constants c, q. We show that 2/3-GAP-qCSP is NP-hard.

Let x be an input for the verifier V. For each of the $2^{c \log n} = n^c$ possible outcomes r of the verifier's random coins, we define a constraint $\phi_{x,r}(\pi) = 1$ if the verifier V accepts proof π on random string r. Each such constraint depends on only the q bits of π that V queries. The collection $\{\phi_{x,r}\}_{r \in \{0,1\}^{c \log n}}$ forms a q-CSP instance.

We analyze the two cases for x:

- If $x \in L$: By the completeness of V, there exists a proof π that is accepted for every r. This assignment satisfies all constraints, so $val(\{\phi_{x,r}\}) = 1$. Thus, $\{\phi_{x,r}\}$ is a YES instance.
- If $x \notin L$: This is the crucial case. By the soundness property of the PCP verifier, for **any** given proof string π, the verifier accepts with probability at most $1/2$. Let $R = \{0,1\}^{c \log n}$ be the set of all possible random strings. The probability of acceptance is the fraction of random strings that cause the verifier to accept:

$$\Pr_{r \in R}[V^\pi(x) = 1] = \frac{|\{r \in R \mid V^\pi(x) \text{ accepts on random string } r\}|}{|R|} \leq \frac{1}{2}$$

By our construction of the CSP instance, an assignment (which is the proof π) satisfies a constraint $\phi_{x,r}$ if and only if $V^\pi(x)$ accepts on random string r. Therefore, the fraction of satisfied constraints for a given assignment π is:

$$\frac{|\{r \in R \mid \phi_{x,r}(\pi) \text{ is satisfied}\}|}{|R|} = \frac{|\{r \in R \mid V^\pi(x) \text{ accepts on random string } r\}|}{|R|}$$

Combining these two statements, we have that for any assignment π:

$$\frac{|\{r \in R \mid \phi_{x,r}(\pi) \text{ is satisfied}\}|}{|R|} \leq \frac{1}{2}$$

The value of the CSP instance, $val(\{\phi_{x,r}\})$, is the *maximum* fraction of satisfiable clauses over *all possible* assignments. Since the fraction is at most $1/2$ for every assignment, the maximum must also be at most $1/2$:

$$val(\{\phi_{x,r}\}) = \max_{\pi} \left(\frac{|\{r \in R \mid \phi_{x,r}(\pi) \text{ is satisfied}\}|}{|R|} \right) \leq \frac{1}{2} < \frac{2}{3}$$

Thus, $\{\phi_{x,r}\}$ is a NO instance of 2/3-GAP-qCSP.

This construction is a polynomial-time reduction from L to 1/2-GAP-qCSP, proving that the latter is NP-hard.

Proof of (2 $\implies$ 1) Assume that for some constants $q, \rho < 1$, the problem ρ-GAP-qCSP is NP-hard. This implies that for any language $L \in \mathbf{NP}$, there exists a polynomial-time reduction function f that maps an input string x to a q-CSP instance $\phi_x = f(x)$.

We use this reduction to construct a PCP verifier V for the language L. The expected PCP proof π is an assignment to the variables of the CSP instance ϕ_x. The verifier V operates as follows on an input x of length n:

1. It first runs the polynomial-time reduction f to compute the q-CSP instance $\phi_x = f(x)$. Let this instance have m constraints.
2. It uses its random bits to select an index $i \in \{1, \ldots, m\}$ uniformly at random.
3. It makes q queries to the proof π to read the values of the variables involved in the chosen constraint ϕ_i.
4. It accepts if and only if the queried values satisfy the constraint ϕ_i.

Analysis of Completeness and Soundness

- **Completeness:** If $x \in L$, then by the property of the reduction, $val(\phi_x) = 1$. This means a satisfying assignment π exists. With this correct proof, every constraint is satisfied, so the verifier will accept with probability 1 regardless of which constraint it chooses.
- **Soundness:** If $x \notin L$, then $val(\phi_x) < \rho$. This means that for any purported proof π, at most a ρ fraction of the constraints are satisfied. The probability that the verifier chooses a satisfied constraint is therefore at most ρ. The soundness error is $\rho < 1$, which can be amplified to $1/2$ by repeating the test a constant number of times.

Analysis of Resource Usage We must show that the verifier uses logarithmic randomness and constant queries.

- **Query Complexity:** By construction, the verifier queries exactly the q variables associated with a single constraint. Since q is a fixed constant, the query complexity is $O(1)$.
- **Randomness:** The verifier needs to select an index from $\{1, \ldots, m\}$ uniformly at random. To specify such an index, it requires $\lceil \log_2 m \rceil$ random bits. We need to show that this quantity is logarithmic in n. Since f is a polynomial-time reduction, the time to compute the instance ϕ_x is bounded by a polynomial in n, say $O(n^c)$ for some constant c. A Turing Machine that runs for T steps can produce an output of size at most T. Therefore, the size of the description of ϕ_x, including all its m constraints, is also bounded by a polynomial in n. This directly implies that the number of constraints m is at most polynomial in n:

$$m \leq O(n^k) \text{ for some constant } k$$

The number of random bits required, $r(n)$, is therefore:

$$r(n) = \lceil \log_2 m \rceil \leq \lceil \log_2(O(n^k)) \rceil = O(\log(n^k)) = O(k \log n) = O(\log n)$$

We have constructed a verifier for L that uses $O(\log n)$ random bits and $O(1)$ queries. Thus, if ρ-GAP-qCSP is NP-hard, then $L \in \mathbf{PCP}(\log n, 1)$. This holds for any $L \in \mathbf{NP}$, so $\mathbf{NP} \subseteq \mathbf{PCP}(\log n, 1)$. □

2.3.4 Equivalence of Gap Problems

Theorem 2.15 (Equivalence of Gap-CSP and Gap-3SAT)**.** *There exist constants $q \in \mathbb{N}$ and $\rho < 1$ such that ρ-GAP-qCSP is NP-hard if and only if there exists a constant $\rho' < 1$ such that ρ'-GAP-3SAT is NP-hard.*

Proof. We prove each direction of the equivalence separately.

Proof of ($\Longleftarrow$): Hardness of Gap-3SAT $\implies$ Hardness of Gap-CSP Assume that for some constant $\rho' < 1$, the problem ρ'-GAP-3SAT is NP-hard. Then, for every language $L \in \mathbf{NP}$, there exists a polynomial-time reduction f_L such that for all inputs x:

$$x \in L \Rightarrow \text{val}(f_L(x)) = 1, \quad x \notin L \Rightarrow \text{val}(f_L(x)) < \rho'.$$

A 3-CNF formula is a special case of a 3-CSP instance—each clause corresponds to a constraint on at most three Boolean variables. Hence, every 3SAT instance can be viewed as a valid 3CSP instance.

Therefore, the same reduction f_L also serves as a polynomial-time reduction from L to ρ'-GAP-3CSP. Since this holds for all $L \in \mathbf{NP}$, it follows that ρ'-GAP-3CSP is NP-hard.

By setting $q = 3$ and $\rho = \rho'$, we conclude that the NP-hardness of ρ'-GAP-3SAT implies the NP-hardness of ρ-GAP-qCSP.

Proof of ($\Longrightarrow$): Hardness of Gap-CSP $\implies$ Hardness of Gap-3SAT Assume that for some constants $q \in \mathbb{N}$ and $\rho < 1$, the problem ρ-GAP-qCSP is NP-hard. We will show a polynomial-time reduction that converts any instance ϕ of q-CSP into a 3-CNF formula ϕ''. We will prove that if an assignment fails to satisfy a certain fraction of constraints in ϕ, then any corresponding assignment for ϕ'' must fail to satisfy a related, non-zero fraction of clauses. This will prove that a corresponding Gap-3SAT problem is also NP-hard.

The reduction has two main steps:

1. Convert each q-CSP constraint into an equivalent CNF formula.
2. Convert the resulting CNF formula into an equisatisfiable 3-CNF formula.

Step 1: From q-CSP to CNF. We first state a necessary lemma.

Lemma 2.16 (Universality of CNF)**.** *Any Boolean function $f : \{0,1\}^k \to \{0,1\}$ can be expressed as a CNF formula ψ_f. The size of ψ_f is at most $k \cdot 2^k$.*

Proof. For any input vector $v \in \{0,1\}^k$ where $f(v) = 0$, we can construct a clause K_v that is false only on input v. For example, if $v = (1,0,1)$, the clause is $(\neg z_1 \vee z_2 \vee \neg z_3)$. The final formula is the conjunction of all such clauses: $\psi_f = \bigwedge_{v:f(v)=0} K_v$. This formula is true if and only if f is true. □

Let ϕ be a q-CSP instance with m constraints, $C_1, \ldots, C_m$. Each constraint C_i is a Boolean function on at most q variables. Using the lemma, we replace each C_i with an equivalent CNF formula ψ_i. Since q is a constant, each ψ_i consists of at most 2^q clauses, and each clause has at

most q literals. Let ϕ' be the conjunction of all these new CNF formulas: $\phi' = \psi_1 \wedge \cdots \wedge \psi_m$. An assignment satisfies C_i if and only if it satisfies all clauses in ψ_i.

Step 2: From CNF to 3-CNF. The formula ϕ' is a CNF formula where each clause has at most q literals. We use the standard technique to reduce each clause of size $k > 3$ into an equisatisfiable set of 3-CNF clauses by introducing $k-3$ new "dummy" variables. This transforms a single clause of size $k \leq q$ into $k-2$ clauses of size 3.

Let the final 3-CNF formula be ϕ''.

Analysis of Polynomial Time The reduction from ϕ to ϕ'' is polynomial-time in the size of the original instance ϕ. The size of ϕ is proportional to m, the number of constraints.

1. In Step 1, for each of the m constraints, we generate a CNF formula. Since q is a constant, the size of this formula (at most $q \cdot 2^q$) is also a constant. This step takes $O(m)$ time.
2. In Step 2, we process each clause of ϕ'. The number of clauses in ϕ' is at most $m \cdot 2^q$. For each clause (of constant size at most q), we perform a constant number of operations to convert it to 3-CNF. This step also takes $O(m)$ time.

Thus, the entire reduction is completed in polynomial time in m.

Analysis of the Value

- **Completeness:** If $val(\phi) = 1$, there exists a satisfying assignment for ϕ. This assignment satisfies every constraint C_i. By our construction, this same assignment (extended appropriately for the dummy variables) will satisfy every sub-formula ψ_i and subsequently every clause in the final formula ϕ''. Therefore, if $val(\phi) = 1$, then $val(\phi'') = 1$.
- **Soundness:** Suppose $val(\phi) < \rho$. Let any assignment for the variables of ϕ'' be given. This induces an assignment for the original variables of ϕ. Let $\epsilon = 1 - val(\phi) > 1 - \rho$. By definition, any assignment violates at least an ϵ fraction of the original constraints.
  - Let m be the number of constraints in ϕ. The number of violated constraints is at least ϵm.
  - For each of these ϵm violated constraints C_i, its corresponding CNF formula ψ_i must be false. This implies at least one clause within ψ_i must be violated.
  - When a clause is violated, the 3-CNF reduction ensures that at least one of its resulting 3-clauses is also violated.
  - Therefore, the number of violated clauses in the final formula ϕ'' is at least ϵm.

  Let m'' be the total number of clauses in ϕ''. The number of clauses in each ψ_i is at most 2^q, and each of those is converted into at most $q-2$ clauses of size 3. Thus, $m'' \leq m \cdot 2^q \cdot (q-2)$. The fraction of violated clauses in ϕ'' is at least $\frac{\epsilon m}{m''}$.

$$\frac{\text{violated clauses in } \phi''}{\text{total clauses in } \phi''} \geq \frac{\epsilon m}{m \cdot 2^q \cdot (q-2)} = \frac{\epsilon}{q \cdot 2^q}$$

  The fraction of *satisfied* clauses in ϕ'' is therefore bounded above:

$$val(\phi'') \leq 1 - \frac{\epsilon}{q \cdot 2^q}$$

  Since $\epsilon > 1 - \rho$, we have:

$$val(\phi'') < 1 - \frac{1-\rho}{q \cdot 2^q}$$

  Let $\rho' = 1 - \frac{1-\rho}{q \cdot 2^q}$. Since q is a constant and $\rho < 1$, it follows that ρ' is also a constant strictly less than 1.

We have shown a reduction from ρ-GAP-qCSP to ρ'-GAP-3SAT. This proves that if the former is NP-hard, the latter must be as well. □

Part II
Proof: Irit Dinur's Gap Amplification

3 Constraint Graphs and Expanders

Irit Dinur's combinatorial proof of the PCP Theorem operates on a specific type of graph structure known as constraint graphs.

3.1 Constraint Graphs

We focus on systems where constraints involve only two variables at a time. This structure is naturally represented by a graph where vertices are variables and edges carry the constraints.

Definition 3.1 (Constraint Graph)**.** *A **constraint graph** is a tuple $G = \langle (V, E), \Sigma, \mathcal{C} \rangle$, where:*

1. *(V, E) is an undirected graph, called the **underlying graph** of G. Vertices may have self-loops, and multiple edges between vertices are allowed.*
2. *V is the set of variables, each taking values from a finite **alphabet** Σ.*
3. *$\mathcal{C} = \{c^e\}_{e \in E}$ is a collection of **constraints**, where each edge $e = (u, v) \in E$ carries a constraint $c^e : \Sigma^2 \to \{TRUE, FALSE\}$.*

An **assignment** is a mapping $\sigma : V \to \Sigma$.

Definition 3.2 (Satisfiability Fraction and Gap)**.** *For a constraint graph G and an assignment σ:*

- $$SAT_\sigma(G) = \Pr_{(u,v)=e \in E}[c^e(\sigma(u), \sigma(v)) = TRUE] = \frac{|\{e = (u, v) \in E \mid c^e(\sigma(u), \sigma(v)) = TRUE\}|}{|E|}$$
- $$SAT(G) = \max_\sigma SAT_\sigma(G)$$
- $$\overline{SAT}_\sigma(G) = 1 - SAT_\sigma(G)$$

  .
- *The **satisfiability-gap** (or simply the **gap**), is defined as:*

$$\overline{SAT}(G) = \min_\sigma \overline{SAT}_\sigma(G) = 1 - SAT(G)$$

An instance G is satisfiable if and only if $SAT(G) = 1$, which is equivalent to $\overline{SAT}(G) = 0$. If G is unsatisfiable, then $\overline{SAT}(G) \geq 1/|E|$.

Even for these restricted 2-variable constraints, determining satisfiability remains hard.

Theorem 3.3 (Constraint-Graph Satisfiability)**.** *Given a constraint graph $G = \langle (V, E), \Sigma, \mathcal{C} \rangle$ with $|\Sigma| = 7$, it is NP-hard to decide if $SAT(G) = 1$.*

Proof. We will prove this by providing a polynomial-time reduction from 3SAT, which is known to be NP-complete.

Given a 3-CNF formula ϕ with m clauses, $C_1, \ldots, C_m$, over n variables, we construct an equivalent constraint graph $G_\phi = \langle (V, E), \Sigma, \mathcal{C} \rangle$ such that $SAT(G_\phi) = 1$ if and only if ϕ is satisfiable.

Construction of the Graph

- **Alphabet Σ:** A 3-CNF clause C_i depends on at most 3 variables. There are $2^3 = 8$ possible assignments to these variables. C_i is unsatisfied by exactly one assignment. We define our alphabet Σ to be the set of the $2^3 - 1 = 7$ assignments that *satisfy* the clause. Thus, $|\Sigma| = 7$.
- **Vertices V:** We create one vertex v_i for each clause C_i in ϕ.
- **Edges E and Constraints $\mathcal{C}$:** We add an edge $e = (v_i, v_j)$ between any two vertices v_i and v_j if their corresponding clauses C_i and C_j share one or more variables. For each such edge, we define a constraint $c^e(\sigma_i, \sigma_j)$ that returns TRUE if and only if the partial assignments represented by $\sigma_i \in \Sigma$ and $\sigma_j \in \Sigma$ are **consistent**. That is, they agree on the truth values of all shared variables.

This reduction is clearly polynomial-time. The number of vertices is m. The number of edges is at most $\binom{m}{2} = O(m^2)$. The constraints are of constant size.

Analysis of Correctness We must show that ϕ is satisfiable if and only if $SAT(G_\phi) = 1$.

ϕ **is satisfiable** $\implies SAT(G_\phi) = 1$ Assume ϕ is satisfiable, and let A be a global satisfying assignment for all n variables.

For each vertex v_i, define its assignment $\sigma(v_i)$ to be the partial assignment corresponding to the restriction of A to the variables in clause C_i. Since A is a satisfying assignment for ϕ, it must satisfy C_i. Therefore, this partial assignment is one of the 7 satisfying assignments in Σ, and $\sigma(v_i)$ is a valid symbol.

Now, consider any constraint c^e on an edge $e = (v_i, v_j)$. Since both $\sigma(v_i)$ and $\sigma(v_j)$ are derived from the *same* global assignment A, their partial assignments must agree on all variables they share. Thus, the constraint c^e is satisfied. Since this holds for all edges, all constraints are satisfied, and $SAT(G_\phi) = 1$.

$SAT(G_\phi) = 1 \implies \phi$ **is satisfiable** Assume $SAT(G_\phi) = 1$. This means there exists an assignment $\sigma : V \to \Sigma$ that satisfies every constraint in $\mathcal{C}$. Build a global assignment A by assigning the truth values to each variable according to the $\sigma(v_i)s$. The assignment is guaranteed to be consistent by definition of the constraint. For any clause C_i, the global assignment A, when restricted to the variables in C_i, is exactly the partial assignment $\sigma(v_i)$. By our construction of Σ, every $\sigma(v_i)$ is an assignment that *satisfies* its corresponding clause C_i. Therefore, A satisfies all clauses, and ϕ is satisfiable.

Since we have shown a polynomial-time reduction, the problem of deciding $SAT(G) = 1$ is NP-hard. □

Definition 3.4 (Size of a Constraint Graph)**.**

$$size(G) = |V| + |E|$$

3.2 Expander Graphs

The gap amplification step in Dinur's proof relies on the underlying constraint graph having certain properties. This is achieved by ensuring the graph is an expander.

Definition 3.5 (Edge Expansion)**.** *Let $G = (V, E)$ be a d-regular graph. For any subset $S \subseteq V$, let $E(S, \overline{S}) = \{(u, v) \in E \mid u \in S, v \notin S\}$. The* ***edge expansion*** *$\phi(G)$ is defined as:*

$$\phi(G) = \min_{S \subset V, 0 < |S| \leq |V|/2} \frac{|E(S, \overline{S})|}{d|S|}$$

A graph is called an expander if

$$\phi(G) = \Omega(1)$$

.

Crucially for our proof, such graphs exist and can be constructed efficiently.

Lemma 3.6 (Existence of Expanders)**.** *There exist constants $d_0 \in \mathbb{N}$ and $\alpha > 0$, such that there is a polynomial-time constructible family $\{X_n\}_{n\in\mathbb{N}}$ of d_0-regular graphs X_n on n vertices with $\phi(X_n) \geq \alpha$.*

Proof. The proof of this lemma is a foundational and non-trivial result in theoretical computer science, and we refer to the original construction, Reingold, Vadhan, and Wigderson [RVW], for a complete proof. □

Lemma 3.7. *Let G be a d-regular graph, and let $\phi(G)$ denote its edge expansion. Let $\lambda_2(G)$ be the second largest eigenvalue of the adjacency matrix of G. The following relation is known:*

$$\lambda_2(G) \leq d(1 - \frac{\phi(G)^2}{2})$$

Proof. From Cheeger's inequality for regular graphs, we have:

$$\phi(G) \leq \sqrt{2 \cdot \sigma_2}$$

where σ_2 is the second smallest eigenvalue of the normalized Laplacian matrix of G. Assume σ is an eigenvalue of the normalized Laplacian L and x is the corresponding eigenvector. Then we have:

$$Lx = \sigma x$$

$$(I - D^{-1/2}AD^{-1/2})x = \sigma x$$
$$(I - A/d)x = \sigma x \quad \text{as } G \text{ is } d\text{-regular}$$
$$Ax = (1 - \sigma)dx$$

Thus,

$$\lambda_2(G) = (1 - \sigma_2)d$$
$$\Rightarrow \sigma_2 = 1 - \frac{\lambda_2(G)}{d}$$

Combining this with Cheeger's inequality, we get:

$$\phi(G) \leq \sqrt{2\left(1 - \frac{\lambda_2(G)}{d}\right)}$$

$$\Rightarrow \phi(G)^2 \leq 2\left(1 - \frac{\lambda_2(G)}{d}\right)$$

$$\Rightarrow \frac{\lambda_2(G)}{d} \leq 1 - \frac{\phi(G)^2}{2}$$

□

This spectral gap implies that random walks on expander graphs "mix" rapidly, meaning they quickly approach a uniform distribution. The following proposition quantifies this behavior, showing that a random walk is unlikely to stay within a small subset of edges.

Theorem 3.8 (Random Walk Estimate)**.** *Let $G = (V, E)$ be a d-regular graph with $\lambda(G)$ as its second largest adjacency matrix eigenvalue. Let $F \subseteq E$ be a set of edges without self loops. The probability p that a random walk, starting at a random edge in F, also has its $(i+1)$-th step in F is bounded by:*

$$p \leq \frac{|F|}{|E|} + \left(\frac{|\lambda(G)|}{d}\right)^i$$

Proof. Let A be the adjacency matrix of G, and let $\mathcal{A} = \frac{1}{d}A$ be its **normalized adjacency matrix**. The eigenvalues of A are $d = \lambda_0 \geq \lambda_1 \geq \cdots \geq \lambda_{n-1}$. Let $\tilde{\lambda} = \frac{\lambda_2(G)}{d}$ be the second largest eigenvalue of $\mathcal{A}$ in absolute value.

We will analyze the probability p by defining two vectors, x and y, and expressing p as an inner product.

1. Vector Definitions Let K be the probability distribution on V induced by picking a random edge $(u, v) \in F$ and then picking one of its two endpoints uniformly at random.

- Let x be the vector representing this initial distribution. The entry $x_v = \Pr_K[v]$ is the probability that a vertex v is selected. This is equal to the fraction of edges in F incident to v, divided by the total number of such endpoints, $2|F|$.

  $$x_v = \frac{\deg_F(v)}{2|F|}$$

  Note that $\sum_v x_v = 1$, so x is a valid probability distribution.

- Let y be a vector where y_v is the probability that a random walk *starting from* v takes a step along an edge in F.

  $$y_v = \frac{\deg_F(v)}{d}$$

2. Expressing p as an Inner Product The probability p is the event that a random walk of length $i+1$ has its $(i+1)$-th step in F, given that it started with the distribution K. Let $\mu_i = \mathcal{A}^i x$ be the probability distribution of the walk at step i. The probability of then taking a step in F from v is y_v. Thus, $p = \sum_{v \in V} (\mu_i)_v \cdot y_v = \langle y, \mu_i \rangle = \langle y, \mathcal{A}^i x \rangle$.

3. Decomposing the Probability The vector x can be decomposed into its component parallel to the stationary distribution and its component orthogonal to it. The stationary distribution for a d-regular graph is the uniform vector $u = \frac{1}{n}\mathbf{1}$ (where

$$\mathcal{A}u = u$$

).

$$x = x^{||} + x^{\perp} \quad \text{where} \quad x^{||} = \frac{\langle x, u \rangle}{\langle u, u \rangle} u = \frac{1}{n}\mathbf{1}$$

We can now split the inner product:

$$p = \langle y, \mathcal{A}^i x^{||} \rangle + \langle y, \mathcal{A}^i x^{\perp} \rangle$$

4. The Stationary Component

$$\langle y, \mathcal{A}^i x^{||} \rangle = \langle y, x^{||} \rangle = \left\langle y, \frac{1}{n}\mathbf{1} \right\rangle = \frac{1}{n} \sum_{v \in V} y_v$$

Substituting the definition of y_v:

$$\frac{1}{n}\sum_{v\in V}\frac{\deg_F(v)}{d} = \frac{1}{nd}\sum_{v\in V}\deg_F(v)$$

The sum of degrees of vertices in F is $2|F|$ (since F has no self-loops). The total number of edges in G is $|E| = \frac{nd}{2}$.

$$\frac{1}{nd}(2|F|) = \frac{2|F|}{nd} = \frac{|F|}{nd/2} = \frac{|F|}{|E|}$$

This first term is the steady-state probability.

5. The Error Component We must now bound the second term, $|\langle y, \mathcal{A}^i x^\perp\rangle|$. We use the Cauchy-Schwarz inequality:

$$|\langle y, \mathcal{A}^i x^\perp\rangle| \le ||y||_2 \cdot ||\mathcal{A}^i x^\perp||_2$$

Since $x^\perp \perp \mathbf{1}$:

$$||\mathcal{A}^i x^\perp||_2 \le \tilde{\lambda}^i ||x^\perp||_2$$

We can bound $||x^\perp||_2 \le ||x||_2$. Now we bound the L_2-norms of x and y:

$$||x||_2^2 = \sum_v x_v^2 = \sum_v \left(\frac{\deg_F(v)}{2|F|}\right)^2 = \frac{1}{4|F|^2}\sum_v (\deg_F(v))^2$$

$$||y||_2^2 = \sum_v y_v^2 = \sum_v \left(\frac{\deg_F(v)}{d}\right)^2 = \frac{1}{d^2}\sum_v (\deg_F(v))^2$$

Since $\deg_F(v) \le d$:

$$||x||_2^2 = \sum_v \frac{(\deg_F(v))^2}{4|F|^2} \le \sum_v \frac{d\cdot\deg_F(v)}{4|F|^2} = \frac{d}{4|F|^2}\sum_v \deg_F(v) = \frac{d\cdot(2|F|)}{4|F|^2} = \frac{d}{2|F|}$$

$$||y||_2^2 = \frac{2|F|}{d}$$

Combining these bounds:

$$||y||_2 \cdot ||x^\perp||_2 \le ||y||_2 \cdot ||x||_2 \le \sqrt{\frac{2|F|}{d}} \cdot \sqrt{\frac{d}{2|F|}} = 1$$

Substituting this back into our Cauchy-Schwarz inequality:

$$|\langle y, \mathcal{A}^i x^\perp\rangle| \le ||y||_2 \cdot ||\mathcal{A}^i x^\perp||_2 \le (||y||_2 \cdot ||x^\perp||_2) \cdot \tilde{\lambda}^i \le 1 \cdot \tilde{\lambda}^i = \tilde{\lambda}^i$$

6. Conclusion We combine the two parts:

$$p = \langle y, \mathcal{A}^i x^{||}\rangle + \langle y, \mathcal{A}^i x^\perp\rangle \le \frac{|F|}{|E|} + |\langle y, \mathcal{A}^i x^\perp\rangle| \le \frac{|F|}{|E|} + \tilde{\lambda}^i$$

Substituting $\tilde{\lambda} = \frac{\lambda(G)}{d}$, we get the final result:

$$p \le \frac{|F|}{|E|} + \left(\frac{|\lambda(G)|}{d}\right)^i$$

□

4 Gap Amplification Operations

The proof of the PCP Theorem is achieved by iteratively applying a "gap amplification" step to a constraint graph. This step is composed of three main operations, each with a specific goal. We will

treat the properties of each operation as a lemma, which will be proved in a later section.

4.1 Step 1: Preprocessing

The first operation, Preprocessing, is a preparatory step. It transforms an arbitrary constraint graph G into a "nicely structured" graph G' that is d-regular and an expander. This structure is required for the subsequent amplification step to work.

Lemma 4.1 (Preprocessing Lemma)**.** *There exist constants $0 < \lambda < d$ and $\beta_1 > 0$ such that any constraint graph G can be transformed in polynomial time into a constraint graph $G' = prep(G)$ with the following properties:*

- *G' is d-regular, has a self-loop on every vertex, $\lambda_2(G') \leq \lambda < d$.*
- *G' has the same alphabet as G, and $size(G') = O(size(G))$.*
- *$\beta_1 \cdot \overline{SAT}(G) \leq \overline{SAT}(G') \leq \overline{SAT}(G)$.*

Proof. The proof of this lemma is deferred to a later Section. □

4.2 Step 2: Powering

The second operation, **Graph Powering**, is the heart of the amplification. It defines a new graph G^t based on t-step walks in the original graph G. This operation amplifies the gap $\overline{SAT}(G)$, but at the cost of increasing the alphabet size.

Definition 4.2 (Graph Powering)**.** *Let $G = \langle (V, E), \Sigma, \mathcal{C} \rangle$ be a d-regular constraint graph, and let $t \in \mathbb{N}$ be a constant. The **powered graph** G^t is a new constraint graph $G^t = \langle (V, E'), \Sigma', \mathcal{C}' \rangle$ defined as follows:*

- ***Vertices** V**:** The vertex set is unchanged, V.*
- ***Alphabet** Σ'**:** The new alphabet is $\Sigma' = \Sigma^{d^{\lceil t/2 \rceil}}$.*

$$\Gamma(u) = \{u' \in V \mid \exists \text{ a path } (u_0, \ldots, u_k) \text{ s.t. } k = \lceil t/2 \rceil, u_0 = u, u_k = u'\}$$

$$|\Gamma(u)| \leq d^{\lceil t/2 \rceil}, \text{ so we pad any incomplete strings if necessary.}$$

  By choosing a canonical ordering, a value $a \in \Sigma'$ for a vertex u is interpreted as a function (an assignment) $a : \Gamma(u) \to \Sigma$. This represents u's "opinion" of the values for all vertices in its $\lceil t/2 \rceil$-step neighborhood.

- ***Edges** E'**:** The edges of G^t correspond to t-step walks in G. For every t-step walk $(u_0, \ldots, u_t)$ in G, we add an edge $e' = (u_0, u_t)$ to E'. This forms a multiset of edges.*
- ***Constraints** $\mathcal{C}'$**:** Let $e' = (u, v)$ be an edge in E' (corresponding to a t-walk). A constraint $c^{e'}(a, b)$ for $a, b \in \Sigma'$ is defined as **TRUE** if and only if the assignments $a : \Gamma(u) \to \Sigma$ and $b : \Gamma(v) \to \Sigma$ are "locally consistent". Formally, $c^{e'}(a, b) = TRUE$ if there exists an assignment $\sigma : \Gamma(u) \cup \Gamma(v) \to \Sigma$ such that:*
  1. *σ is consistent with a on $\Gamma(u)$ (i.e., $\forall w \in \Gamma(u), \sigma(w) = a(w)$).*
  2. *σ is consistent with b on $\Gamma(v)$ (i.e., $\forall w \in \Gamma(v), \sigma(w) = b(w)$).*
  3. *σ satisfies all original constraints from G that lie between the two neighborhoods. That is, for every edge $(w_1, w_2) \in E$ such that $w_1 \in \Gamma(u)$ and $w_2 \in \Gamma(v)$, the constraint $c^{(w_1, w_2)}(\sigma(w_1), \sigma(w_2))$ is TRUE.*

This operation, when applied to a preprocessed expander graph, achieves the desired amplification.

Lemma 4.3 (Amplification Lemma)**.** *Let G be a d-regular constraint graph with self-loops and $\lambda_2(G) \leq \lambda < d$. There exists a constant $\beta_2 > 0$ (depending on $\lambda, d, |\Sigma|$) such that for any $t \in \mathbb{N}$, the powered graph G^t satisfies:*

- *(Completeness) If $\overline{SAT}(G) = 0$, then $\overline{SAT}(G^t) = 0$.*
- *(Soundness) $\overline{SAT}(G^t) \geq \beta_2\sqrt{t} \cdot \min\left(\overline{SAT}(G), \frac{1}{t}\right)$.*

Proof. The proof of this lemma is deferred to a later Section. □

4.3 Step 3: Composition

The "Powering" operation successfully amplifies the gap, but at the cost of increasing the alphabet size. To allow for iteration, we must reduce the alphabet back to a constant size.

This alphabet reduction is achieved by **Composition**. This transformation is based on a tool called an Assignment Tester, $\mathcal{P}$.

Definition 4.4 (Relative Hamming Distance)**.** *Let Σ be a finite alphabet. For any two strings $x, y \in \Sigma^n$ of the same length n, the **Hamming distance**, denoted $dist(x, y)$, is the number of positions at which they differ:*

$$dist(x, y) = |\{i \in \{1, \ldots, n\} \mid x_i \neq y_i\}|$$

*The **relative distance**, denoted $rdist(x, y)$, is the Hamming distance normalized by the string length:*

$$rdist(x, y) = \frac{dist(x, y)}{n}$$

When comparing an assignment $a \in \Sigma^n$ to a set of assignments $S \subseteq \Sigma^n$, $rdist(a, S)$ denotes the minimum relative distance from a to any element in S:

$$rdist(a, S) = \min_{s \in S}\{rdist(a, s)\}$$

Definition 4.5 (Assignment Tester)**.** *An **Assignment Tester** $\mathcal{P}$ with alphabet Σ_0 and rejection probability $\epsilon > 0$ is an algorithm that takes a Boolean circuit Φ (logical function over a set of Boolean variables X) as input, and outputs a constraint graph $G_\Phi = \langle (V, E), \Sigma_0, \mathcal{C} \rangle$ such that:*

- ***Variable Containment:** The new vertex set V contains the original variables X, i.e., $X \subseteq V$.*
- ***Completeness:** If an assignment $a : X \to \{0, 1\}$ satisfies Φ (denoted $a \in SAT(\Phi)$, where $SAT(\phi)$ is the set all assignments that satisfy Φ), then there exists an extension $b : (V \setminus X) \to \Sigma_0$ such that the combined assignment $a \cup b$ satisfies the entire graph:*

$$\overline{SAT}_{a \cup b}(G_\Phi) = 0$$

- ***Soundness:** If $a \notin SAT(\Phi)$, then for **any** extension b, the assignment $a \cup b$ fails to satisfy a fraction of constraints proportional to its distance from being valid.:*

$$\overline{SAT}_{a \cup b}(G_\Phi) \geq \epsilon \cdot rdist(a, SAT(\Phi))$$

Theorem 4.6 (Existence of Assignment Tester)**.** *There is an explicit construction of an assignment tester $\mathcal{P}$ with a constant alphabet Σ_0 and a constant rejection probability $\epsilon > 0$.*

Proof. The proof of this theorem, which typically uses a tool called the Long-Code test, is omitted for brevity. □

Using this tool, we now formally define the "Composition Operation" (denoted by $\circ\mathcal{P}$). This operation first converts the large-alphabet constraints into Boolean circuits and then applies the tester $\mathcal{P}$ to them.

Definition 4.7 (Composition Operation ($\circ\mathcal{P}$))**.** *Let $G = \langle (V, E), \Sigma', \mathcal{C} \rangle$ be a constraint graph. Let $\mathcal{P}$ be an assignment tester (with alphabet Σ_0) and let $Enc : \Sigma' \to \{0,1\}^l$ be an encoding function.*

The new constraint graph $G' = G \circ \mathcal{P} = \langle (V', E'), \Sigma_0, \mathcal{C}' \rangle$ is defined in two steps:

1. ***Robustization:*** *First, we convert each constraint $c^e \in \mathcal{C}$ to a Boolean circuit Φ_e as follows.*
   - *For each original variable $v \in V$, let $[v]$ be a fresh set of l Boolean variables.*
   - *For each edge $e = (u, v) \in E$, the circuit Φ_e will be a circuit on the $2l$ Boolean input variables $[u] \cup [v]$.*
   - *The logic of the circuit Φ_e is defined as:*

$$\Phi_e([u],[v]) = 1 \iff \exists a_u, a_v \in \Sigma' \text{ s.t. } \begin{cases} [u] = Enc(a_u) \\ [v] = Enc(a_v) \\ c^e(a_u, a_v) = TRUE \end{cases}$$

2. ***Composition:*** *Run the assignment tester $\mathcal{P}$ on each circuit Φ_e.*
   - *Let $G_e = \mathcal{P}(\Phi_e) = \langle (V_e, E_e), \Sigma_0, \mathcal{C}_e \rangle$ denote the resulting constraint graph.*
   - *Recall that by the definition of an assignment tester, V_e contains the input variables of Φ_e, so $[u] \cup [v] \subset V_e$.*
   - *Finally, define the new constraint graph G':*

$$V' = \bigcup_{e \in E} V_e \quad ; \quad E' = \bigcup_{e \in E} E_e \quad ; \quad \mathcal{C}' = \bigcup_{e \in E} \mathcal{C}_e$$

The properties of this new graph G' are summarized by the following lemma.

Lemma 4.8 (Composition Lemma)**.** *Assume the existence of an assignment tester $\mathcal{P}$ with a constant rejection probability $\epsilon > 0$ and a constant-size alphabet Σ_0. There exist constants $\beta_3 > 0$ (depending on $\mathcal{P}$) and $C > 0$ (depending on $\mathcal{P}$ and $|\Sigma|$) such that the following holds.*

Given any constraint graph $G = \langle (V, E), \Sigma, \mathcal{C} \rangle$, one can compute, in time linear in $size(G)$, the constraint graph $G' = G \circ \mathcal{P}$, such that:

- $size(G') \leq C \cdot size(G)$
- $\beta_3 \cdot \overline{SAT}(G) \leq \overline{SAT}(G') \leq \overline{SAT}(G)$

Proof. The proof of this lemma is deferred to a later Section. □

5 Proof of the PCP Theorem via Iteration

We now assemble the three operations from the previous section to prove the main theorem, which provides the iterative step for gap amplification.

5.1 The Main Amplification Theorem

By combining Preprocessing, Powering, and Composition, we get a single transformation $G \to G'$ that doubles the gap, while only incurring a linear blowup in size and returning the alphabet to a constant size.

Theorem 5.1 (Main Amplification Step)**.** *There exists a constant alphabet Σ_0 and constants $C > 0$ and $0 < \alpha < 1$ such that the following holds. Given a constraint graph G (over any alphabet Σ), one can construct, in polynomial time, a new constraint graph $G' = \langle (V', E'), \Sigma_0, \mathcal{C}' \rangle$ such that:*

- $size(G') \leq C \cdot size(G)$.
- *(Completeness) If* $\overline{SAT}(G) = 0$ *then* $\overline{SAT}(G') = 0$.
- *(Soundness)* $\overline{SAT}(G') \geq \min(2 \cdot \overline{SAT}(G), \alpha)$.

Proof. Let G be the input constraint graph. We construct G' by applying the three operations in sequence for a fixed constant t (to be chosen later):

$$G' = (\text{prep}(G))^t \circ \mathcal{P}$$

Let $H_1 = \text{prep}(G)$, $H_2 = (H_1)^t$, and $G' = H_2 \circ \mathcal{P}$. We analyze the properties of this combined transformation by applying our three lemmas.

Size Each step incurs at most a constant-factor blowup in size (since t and the alphabets involved are constants independent of the graph size n):

- $size(G') \leq C_3 \cdot size(H_2)$ (from Composition Lemma, where C_3 is a constant).
- $size(H_2) \leq C_2 \cdot size(H_1)$. From the definition of Powering, the vertex set V is unchanged. The new edges E' in $H_2 = (H_1)^t$ correspond to all t-step walks in H_1. Since H_1 is d-regular (from Preprocessing), the number of t-step walks is $|V| \cdot d^t$. The original number of edges is $|E_1| = |V| \cdot d/2$. The size blowup is $O(d^{t-1})$, which is a constant C_2 since d and t are constants.
- $size(H_1) \leq C_1 \cdot size(G)$ (from Preprocessing Lemma, where C_1 is a constant).

Therefore, $size(G') \leq (C_1 C_2 C_3) \cdot size(G)$. Let $C = C_1 C_2 C_3$, which is a constant. Thus, $size(G') \leq C \cdot size(G)$.

Completeness This property is clearly maintained at each step:

$$\overline{SAT}(G) = 0 \implies \overline{SAT}(H_1) = 0 \implies \overline{SAT}(H_2) = 0 \implies \overline{SAT}(G') = 0$$

Soundness We chain the soundness inequalities from the three lemmas:

1. $\overline{SAT}(G') \geq \beta_3 \cdot \overline{SAT}(H_2)$ (from Composition Lemma)
2. $\overline{SAT}(H_2) \geq \beta_2 \sqrt{t} \cdot \min(\overline{SAT}(H_1), 1/t)$ (from Amplification Lemma)
3. $\overline{SAT}(H_1) \geq \beta_1 \cdot \overline{SAT}(G)$ (from Preprocessing Lemma)

Combining these, we get:

$$\overline{SAT}(G') \geq \beta_3 \cdot \left(\beta_2 \sqrt{t} \cdot \min(\beta_1 \cdot \overline{SAT}(G), 1/t) \right)$$

$$\overline{SAT}(G') \geq (\beta_1 \beta_2 \beta_3 \sqrt{t}) \cdot \min\left(\overline{SAT}(G), \frac{1}{\beta_1 t} \right)$$

Let $K = \beta_1 \beta_2 \beta_3$. We want to choose a fixed constant t large enough such that $K\sqrt{t} \geq 2$. We can pick $t = \lceil (2/K)^2 \rceil$. Since K is a constant (depending only on $\lambda, d, \Sigma, \mathcal{P}$), t is also a fixed constant.

With this choice of t, the inequality becomes:

$$\overline{SAT}(G') \geq 2 \cdot \min\left(\overline{SAT}(G), \frac{1}{\beta_1 t} \right)$$

Let $\alpha = \frac{2}{\beta_1 t}$. Since β_1 and t are constants, α is a constant. This gives the final result:

$$\overline{SAT}(G') \geq \min(2 \cdot \overline{SAT}(G), \alpha)$$

□

5.2 The Final Step: Iterative Application

The main theorem provides a single step that doubles the gap. The full proof of the PCP Theorem (in its gap-hardness form) is achieved by iterating this step $O(\log n)$ times.

Theorem 5.2 (Hardness of Gap-3SAT ≡ PCP Theorem)**.** *There exist constants $\rho < 1$ such that GAP-3SAT is NP-hard.*

Proof. Let $G_0 = \langle (V_0, E_0), \Sigma_0', \mathcal{C}_0 \rangle$ be an instance of the Constraint-Graph Satisfiability problem with alphabet $|\Sigma_0'| = 7$. From our previous proof, it is NP-hard to distinguish between the two following cases:

- **Case 1 (YES):** $\overline{SAT}(G_0) = 0$.
- **Case 2 (NO):** $\overline{SAT}(G_0) > 0$.

As noted before, if G_0 is unsatisfiable (Case 2), the gap must be at least one violated edge: $\overline{SAT}(G_0) \geq 1/|E_0| \geq 1/n$, where $n = size(G_0)$.

We now iteratively apply our 'Main Amplification Step' k times, where $k = \lceil \log_2 n \rceil = O(\log n)$. Let $G_{i+1} = \text{transform}(G_i)$ and $G_f = G_k$ be the final graph.

Alphabet Analysis This is the key step. Our Main Amplification Theorem is designed to take a graph with *any* constant-sized alphabet and produce a graph with a *fixed* constant alphabet Σ_0.

- **Step 1:** We apply the transform to G_0 (with alphabet $|\Sigma_0'| = 7$). The output is G_1, which has the fixed, constant alphabet Σ_0.
- **All other steps:** We apply the transform to G_1 (alphabet Σ_0) to get G_2 (alphabet Σ_0), and so on.

Size The size of the final graph G_f is $size(G_f) \leq C \cdot size(G_{k-1}) \leq \cdots \leq C^k \cdot size(G_0)$. Since C is a constant and $k = O(\log n)$, the final size is:

$$size(G_f) \leq C^{O(\log n)} \cdot n = n^{O(\log C)} \cdot n = n^{O(1)}$$

Since our Main Amplification Step runs in time polynomial in its input size, and the size of every intermediate graph G_i is also bounded by $n^{O(1)}$, the total time for all $k = O(\log n)$ steps remains polynomial in n.

Completeness If G_0 is satisfiable, $\overline{SAT}(G_0) = 0$. By the completeness of the main theorem, $\overline{SAT}(G_1) = 0$, and by induction, $\overline{SAT}(G_f) = 0$.

Soundness If G_0 is unsatisfiable, $\overline{SAT}(G_0) \geq 1/n$. By induction, as long as the gap is less than α, the i-th step gives:

$$\overline{SAT}(G_i) \geq 2^i \cdot \overline{SAT}(G_0)$$

After $k = \lceil \log_2 n \rceil$ steps, the gap will be at least:

$$\overline{SAT}(G_f) \geq \min(2^k \cdot \overline{SAT}(G_0), \alpha)$$

$$\overline{SAT}(G_f) \geq \min(2^{\lceil \log_2 n \rceil} \cdot (1/n), \alpha)$$

$$\overline{SAT}(G_f) \geq \min(n \cdot (1/n), \alpha) = \min(1, \alpha) = \alpha$$

Thus, we have created a polynomial-time reduction that maps an instance G_0 to G_f such that:

- G_0 is satisfiable $\implies \overline{SAT}(G_f) = 0$.
- G_0 is unsatisfiable $\implies \overline{SAT}(G_f) \geq \alpha$.

This proves that it is NP-hard to distinguish this gap in G_f. However, G_f is a constraint graph over the non-Boolean alphabet Σ_0. To complete our proof, we must show a final polynomial-time reduction f_{gadget} that maps G_f to a 3-CNF formula ϕ_{final} that preserves this gap.

Construction of the Reduction f_{gadget} Let G_f be our input graph. The alphabet Σ_0 is constant-sized; let $k = |\Sigma_0|$ and let the symbols be $s_1, \ldots, s_k$. The output 3-CNF formula ϕ_{final} is built from two types of "gadgets":

1. **Variable Gadgets (1-hot encoding):** For each variable $v \in V_f$, we create k new Boolean variables, $Z_v = \{z_v^1, \ldots, z_v^k\}$. The intended meaning is that z_v^i = TRUE if and only if v is assigned the symbol s_i. To enforce this "1-hot" encoding, we add two sets of clauses to ϕ_{final} for each $v \in V_f$:
   - **"At Least One" Clause:** We add one clause ensuring at least one z_v^i is true:
     $$(z_v^1 \vee z_v^2 \vee \cdots \vee z_v^k)$$
     Since k is a constant, this k-CNF clause is converted into an equisatisfiable set of $O(k)$ 3-CNF clauses by introducing $O(k)$ new "dummy" Boolean variables.
   - **"At Most One" Clauses:** For every pair of indices $i \neq j$, we add one 2-CNF clause:
     $$(\neg z_v^i \vee \neg z_v^j)$$

   Let C_V be the set of all clauses from all variable gadgets.
2. **Constraint Gadgets:** For each constraint $c^e \in \mathcal{C}_f$ on an edge $e = (u, v)$:
   - This constraint c^e is just a function $c^e : \Sigma_0 \times \Sigma_0 \to \{\text{TRUE}, \text{FALSE}\}$.
   - For every pair of symbols (s_i, s_j) that *violates* the constraint (i.e., $c^e(s_i, s_j) = \text{FALSE}$), we add one 2-CNF clause to ϕ_{final} that forbids this assignment:
     $$(\neg z_u^i \vee \neg z_v^j)$$
     This clause states "it is not allowed for u to be s_i AND v to be s_j at the same time."

   Let C_E be the set of all clauses from all constraint gadgets.

The final formula is $\phi_{\text{final}} = C_V \wedge C_E$.

Analysis of the Reduction We now prove that f_{gadget} is a valid polynomial-time gap-preserving reduction.

- **Polynomial Time:** Let $n = size(G_f) = |V_f| + |E_f|$.
  - The number of Boolean variables is $O(|V_f| \cdot k) + O(|V_f| \cdot k) = O(n)$.
  - The number of clauses in C_V is $O(|V_f| \cdot (k + \binom{k}{2})) = O(n)$.
  - The number of clauses in C_E is $O(|E_f| \cdot k^2) = O(n)$.

  Since $k = |\Sigma_0|$ is a constant, the size of ϕ_{final} is $O(n)$, and it is constructible in linear time. The reduction is polynomial-time.
- **Completeness:** Assume $\overline{SAT}(G_f) = 0$ then clearly $val(\phi_{\text{final}}) = 1$ by construction.

- **Soundness:** Assume $\overline{SAT}(G_f) \geq \alpha$. Let A be any Boolean assignment for ϕ_{final}. We must show it violates a constant fraction of clauses.

  We construct a "decoded" assignment $\sigma_A : V_f \to \Sigma_0$ from A. For each $v \in V_f$:

  - If A's assignment to Z_v is a valid 1-hot encoding of some symbol s_i, set $\sigma_A(v) = s_i$.
  - If A's assignment to Z_v is invalid (e.g., all FALSE, or multiple TRUEs), set $\sigma_A(v) = s_1$ (an arbitrary default symbol).

  By the soundness of G_f, we know this assignment σ_A must violate at least $m_v = \alpha \cdot |E_f|$ constraints in G_f. Let $E_{\text{viol}} \subseteq E_f$ be this set of m_v violated edges.

  Now, consider an edge $e = (u, v) \in E_{\text{viol}}$. Let $\sigma_A(u) = s_i$ and $\sigma_A(v) = s_j$. Since e is violated, $c^e(s_i, s_j) = \text{FALSE}$.

  - **Case 1: A is "honest" about u and v.** If the assignments to Z_u and Z_v are both valid 1-hot encodings, then $A(z_u^i) = \text{TRUE}$ and $A(z_v^j) = \text{TRUE}$. By construction (Step 2), the clause $(\neg z_u^i \vee \neg z_v^j)$ must be in C_E. This clause is violated by A.
  - **Case 2: A is "dishonest" about u or v.** If the assignment to Z_u (or Z_v) is *not* a valid 1-hot encoding, then A must be violating at least one of the "At Least One" or "At Most One" clauses in C_V for vertex u.

  In either case, each of the m_v violated edges from G_f points to at least one violated clause in ϕ_{final}.

  Let N_{viol} be the total number of clauses violated by A. Let m_c be the number of violated clauses in C_E and m_v be the number of violated variable gadgets in C_V. A single violated variable gadget (e.g., for v) can be incident to at most D edges, where D is the maximum degree of G_f (which is a constant, d^t).

  The $\alpha|E_f|$ violated edges are a subset of (violated constraint gadgets $\cup$ edges incident to violated variable gadgets).

  $$\alpha|E_f| \leq m_c + m_v \cdot D$$

  The total number of violated clauses is $N_{\text{viol}} \geq m_c + m_v$.

  $$N_{\text{viol}} \geq m_v + \frac{\alpha|E_f| - m_v \cdot D}{1} \geq \frac{1}{D}(m_v D + \alpha|E_f| - m_v D) = \frac{\alpha|E_f|}{D}$$

  (This is a loose but sufficient bound). The total number of clauses in ϕ_{final} is $m_{\text{total}} = O(|V_f| + |E_f|)$. Since G_f is constant-degree, $|V_f|$ is proportional to $|E_f|$, so $m_{\text{total}} = O(|E_f|)$.

  The fraction of violated clauses is $\frac{N_{\text{viol}}}{m_{\text{total}}} \geq \frac{\alpha|E_f|/D}{O(|E_f|)} = c'(\alpha)$. Let $\alpha' = c'(\alpha)$. Since α is a constant, α' is also a constant. Thus, $val(\phi_{\text{final}}) \leq 1 - \alpha'$.

Conclusion We have shown a complete chain of polynomial-time reductions:

$$\text{Constraint-Graph Satisfiability} \leq_p G_f \leq_p \text{Gap-3SAT}$$

Since we proved the first problem is NP-hard, by transitivity, 'Gap-3SAT' is NP-hard. □

Part III
Additional Proofs and Conclusion

6 Proofs of Key Lemmas

This section provides the proofs for the three main lemmas that were stated and used in the previous part.

6.1 Proof of Preprocessing

Original Lemma:

Lemma 6.1 (Preprocessing Lemma)**.** *There exist constants $0 < \lambda < d$ and $\beta_1 > 0$ such that any constraint graph G can be transformed in polynomial time into a constraint graph $G' = prep(G)$ with the following properties:*

- *G' is d-regular, has a self-loop on every vertex, $\lambda_2(G') \leq \lambda < d$.*
- *G' has the same alphabet as G, and $size(G') = O(size(G))$.*
- *$\beta_1 \cdot \overline{SAT}(G) \leq \overline{SAT}(G') \leq \overline{SAT}(G)$.*

We achieve this in two steps. First, we make the graph regular. Second, we add edges to make it an expander with self-loops.

6.1.1 Step 1: Regularization

The first step, prep_1, transforms an arbitrary graph into a d-regular graph by replacing each vertex v with new vertices.

Lemma 6.2 (Constant Degree Transformation)**.** *There exist constants $d_0 > 0$ and $\beta_a > 0$ such that any constraint graph $G = \langle (V, E), \Sigma, \mathcal{C} \rangle$ can be transformed into a $(d_0 + 1)$-regular constraint graph $G_1 = \langle (V', E'), \Sigma, \mathcal{C}' \rangle$ such that $size(G_1) = O(size(G))$ and:*

$$\beta_a \cdot \overline{SAT}(G) \leq \overline{SAT}(G_1) \leq \overline{SAT}(G)$$

Proof. We construct $G_1 = \text{prep}_1(G)$ as follows. Let X_n be a d_0-regular expander graph on n vertices, which we know is polynomial-time constructible.

- **Vertices V':** For each vertex $v \in V$ with degree $deg(v)$, we create $deg(v)$ new vertices, which we denote $[v]$. The new vertex set is $V' = \bigcup_{v \in V} [v]$. The total number of new vertices is $|V'| = \sum_{v \in V} deg(v) = 2|E|$.
- **Edges E':** The new edge set $E' = E_{\text{internal}} \cup E_{\text{external}}$ is a union of two types of edges:
  1. E_{internal}: For each $[v]$, we place a d_0-regular expander graph $X_{deg(v)}$ on its vertices.
  2. E_{external}: For each original edge $e = (u, v) \in E$, we pick one vertex from the cloud $[u]$ and one vertex from the cloud $[v]$ and connect them with an edge. We do this in such a way that every vertex in V' is part of exactly one such external edge.
- **Constraints $\mathcal{C}'$:**
  - For all $e' \in E_{\text{internal}}$, we set $c^{e'}(a, b) = \text{TRUE} \iff a = b$.
  - For each $e' = (u', v') \in E_{\text{external}}$ (which corresponds to an original edge $e = (u, v)$), we set $c^{e'} = c^e$.

Size and Regularity By construction, every vertex $v' \in V'$ has degree d_0 (from its internal expander) plus degree 1 (from its single external edge). Thus, G_1 is $(d_0 + 1)$-regular. The new size is $|V'| + |E'| = 2|E| + (d_0 + 1)|E| = O(|E|) = O(size(G))$.

Completeness ($\overline{SAT}(G_1) \leq \overline{SAT}(G)$) Let σ be an assignment for G with $\overline{SAT}_\sigma(G) = \epsilon$. We construct an assignment σ' for G_1 by setting $\sigma'(v') = \sigma(v)$ for all $v' \in [v]$.

- All E_{internal} constraints are satisfied, as all vertices in a cloud get the same value.
- The E_{external} edges violate a constraint if and only if the original edge in G was violated.

The number of violated edges in G_1 is $\epsilon \cdot |E|$. The total number of edges in G_1 is $(d_0+1)|E|$. Thus, $\overline{SAT}(G_1) \leq \overline{SAT}_{\sigma'}(G_1) = \frac{\epsilon|E|}{(d_0+1)|E|} = \frac{\epsilon}{d_0+1} \leq \epsilon = \overline{SAT}_{\sigma}(G)$. Since this holds for all assignments, $\overline{SAT}(G_1) \leq \overline{SAT}(G)$.

Soundness ($\beta_a \cdot \overline{SAT}(G) \leq \overline{SAT}(G_1)$) Let σ' be the optimal assignment for G_1, with $\overline{SAT}(G_1) = \alpha$. We define an assignment σ for G using the "plurality vote" within each cloud:

$$\sigma(v) = \text{most popular value assigned by } \sigma' \text{ to vertices in } [v]$$

Let $F \subseteq E$ be the set of edges in G violated by this plurality assignment σ. By definition, $\overline{SAT}(G) \leq |F|/|E|$. Let $S \subseteq V'$ be the set of "disagreeing" vertices, i.e., $S = \{v' \in [v] \mid \sigma'(v') \neq \sigma(v)\}$.

For any original edge $e = (u, v) \in F$, consider its corresponding external edge $e' = (u', v') \in E_{\text{external}}$. The constraint c^e is violated by $(\sigma(u), \sigma(v))$. If σ' satisfies e', it *must* be because at least one of its endpoints disagrees with the plurality (i.e., $\sigma'(u') \neq \sigma(u)$ or $\sigma'(v') \neq \sigma(v)$). Therefore, either e' is violated by σ', or at least one of its endpoints is in S.

Let $F' \subseteq E'$ be the set of edges in G_1 violated by σ'.

$$|F| \leq |F' \cap E_{\text{external}}| + |S|$$

Now, consider the vertices in S. Within any $[v]$, the set $S \cap [v]$ is a collection of minority assignments. Look at the below argument:

1. **Local View:** The set of disagreeing vertices in [v] is $S^v = S \cap [v]$. We can partition this set S^v into "minority groups" based on their assigned value:

   $$S^v = \bigcup_{a \neq \sigma(v)} S_a \quad \text{where} \quad S_a = \{v' \in S^v \mid \sigma'(v') = a\}$$

   By the definition of $\sigma(v)$ as the plurality value, any such set S_a must be a minority: $|S_a| \leq |[v]|/2$.

2. **Applying the Expander Lemma:** The graph on $[v]$ is a d_0-regular expander $X_{deg(v)}$. The Expander Lemma (Lemma 3.6) states that for any minority set S_a:

   $$|E_v(S_a, \overline{S_a})| \geq h_0 \cdot |S_a|$$

   where h_0 is the associated constant.

3. **Connecting Edges to Violations:** Every edge $e' \in E_v(S_a, \overline{S_a})$ connects a vertex assigned value a to a vertex assigned a value *not* equal to a. By the equality constraint, every single one of these edges is violated by σ' and is therefore in $F' \cap E_{\text{internal}}$.

4. **Summing Over All Clouds:** We can sum this lower bound over all minority sets in all clouds to get the total number of violated internal edges:

   $$|F' \cap E_{\text{internal}}| \geq \sum_{v \in V} \sum_{a \neq \sigma(v)} |E_v(S_a, \overline{S_a})|$$

   $$|F' \cap E_{\text{internal}}| \geq \sum_{v \in V} \sum_{a \neq \sigma(v)} (h_0 \cdot |S_a|)$$

   Factoring out the constant h_0:

   $$|F' \cap E_{\text{internal}}| \geq h_0 \cdot \left(\sum_{v \in V} \sum_{a \neq \sigma(v)} |S_a| \right)$$

The inner sum is just the total number of disagreeing vertices in the cloud $[v]$, which is $|S^v|$. Summing over all clouds v gives the total number of disagreeing vertices in the whole graph, $|S|$:

$$|F' \cap E_{\text{internal}}| \geq h_0 \cdot \left(\sum_{v \in V} |S^v| \right) = h_0 \cdot |S|$$

$$|S| \leq \frac{1}{h_0} |F' \cap E_{\text{internal}}|$$

Combining these, we get:

$$|F| \leq |F' \cap E_{\text{external}}| + \frac{1}{h_0} |F' \cap E_{\text{internal}}| \leq (1 + 1/h_0) \cdot |F'|$$

Now we relate the fractions:

$$\overline{SAT}(G) \leq \frac{|F|}{|E|} \leq \frac{(1 + 1/h_0)|F'|}{|E|}$$

Since $|E'| = (d_0 + 1)|E|$, we have $|E| = |E'|/(d_0 + 1)$.

$$\overline{SAT}(G) \leq \frac{(1 + 1/h_0)|F'|}{|E'|/(d_0 + 1)} = (1 + 1/h_0)(d_0 + 1) \cdot \frac{|F'|}{|E'|} = (1 + 1/h_0)(d_0 + 1) \cdot \overline{SAT}(G_1)$$

By setting $\beta_a = \frac{1}{(1+1/h_0)(d_0+1)}$, which is a positive constant, we have $\beta_a \cdot \overline{SAT}(G) \leq \overline{SAT}(G_1)$. □

6.1.2 Step 2: Expanderizing

The second step, prep_2.

Lemma 6.3 (Expanderizing Transformation)**.** *There exist constants $d'_0 > 0, h'_0 > 0$, and $\beta_b > 0$ such that any d_1-regular constraint graph G_1 can be transformed into a d-regular graph G' ($d = d_1 + d'_0 + 1$) such that G' has self-loops, $\lambda_2(G') \leq d - \frac{(h'_0)^2}{2d}$, and:*

$$\beta_b \cdot \overline{SAT}(G_1) \leq \overline{SAT}(G') \leq \overline{SAT}(G_1)$$

Proof. Let $G_1 = \langle (V', E_1), \Sigma, \mathcal{C}_1 \rangle$ be the d_1-regular graph.

- **New Edges:** We add two sets of edges to V':
  1. E_{expander}: The edges of a d'_0-regular expander $X_{|V'|}$ with $\phi(G) \geq \frac{h_0}{d}$.
  2. E_{loop}: A self-loop (v, v) for every vertex $v \in V'$.
- **Final Graph G':** The new graph is $G' = \langle (V', E'), \Sigma, \mathcal{C}' \rangle$, where:
  - $E' = E_1 \cup E_{\text{expander}} \cup E_{\text{loop}}$ (as a multiset).
  - $\mathcal{C}'$ contains all constraints from $\mathcal{C}_1$ for edges in E_1.
  - For all edges $e' \in E_{\text{expander}} \cup E_{\text{loop}}$, $c^{e'}$ is the **void constraint** (always TRUE).

Size and Regularity The new degree of every vertex is $d = d_1 + d'_0 + 1$, so G' is d-regular. The size is $size(G') = |V'| + |E'| = |V'| + |E_1| + |E_{\text{expander}}| + |E_{\text{loop}}| = O(size(G_1))$.

Eigenvalue By Lemma 3.7 (Cheeger's inequality), we have:

$$\lambda_2(G') \leq d - \frac{(h'_0)^2}{2d}$$

Letting $\lambda = d - \frac{(h'_0)^2}{2d}$, we have $\lambda_2(G') \leq \lambda < d$.

Gap Analysis Let σ be any assignment. The void constraints on E_{expander} and E_{loop} are always satisfied. Therefore, the set of violated edges in G' is *exactly* the set of violated edges in G_1. Let $V_\sigma \subseteq E_1$ be the set of edges violated by σ.

$$\overline{SAT}_\sigma(G_1) = \frac{|V_\sigma|}{|E_1|} \quad ; \quad \overline{SAT}_\sigma(G') = \frac{|V_\sigma|}{|E'|}$$

We know $|E'| = |E_1| + |E_{\text{expander}}| + |E_{\text{loop}}| = |E_1| + \frac{d_0'|V'|}{2} + |V'|$. Since G_1 is d_1-regular, $|E_1| = d_1|V'|/2$.

$$|E'| = \frac{d_1|V'|}{2} + \frac{d_0'|V'|}{2} + |V'| = \frac{(d_1 + d_0' + 2)|V'|}{2} = \frac{d+1}{d_1}|E_1|$$

- **Completeness:** Since $|E'| \geq |E_1|$, we have:

  $$\overline{SAT}_\sigma(G') = \frac{|V_\sigma|}{|E'|} \leq \frac{|V_\sigma|}{|E_1|} = \overline{SAT}_\sigma(G_1)$$

  This holds for the optimal assignment, so $\overline{SAT}(G') \leq \overline{SAT}(G_1)$.

- **Soundness:**

  $$\overline{SAT}_\sigma(G') = \frac{|V_\sigma|}{|E'|} = \frac{|V_\sigma|}{(d + 1/d_1)|E_1|} = \frac{d_1}{d+1} \cdot \frac{|V_\sigma|}{|E_1|} = \frac{d_1}{d+1} \cdot \overline{SAT}_\sigma(G_1)$$

  This holds for any σ, so: Setting $\beta_b = \frac{d_1}{d+1}$ (a positive constant) gives $\beta_b \cdot \overline{SAT}(G_1) \leq \overline{SAT}(G')$.

□

6.1.3 Final Proof of Preprocessing Lemma

The full preprocessing operation is $G' = \text{prep}(G) = \text{prep}_2(\text{prep}_1(G))$.

- **Properties:** The graph G' is d-regular, has self-loops, and has the desired eigenvalue bound $\lambda_2(G') \leq \lambda < d$ (from Lemma 6.3).
- **Size:** $size(G') = O(size(\text{prep}_1(G))) = O(size(G))$.
- **Gap:** We combine the gap inequalities:

  $$\overline{SAT}(G') \leq \overline{SAT}(\text{prep}_1(G)) \leq \overline{SAT}(G)$$

  $$\overline{SAT}(G') \geq \beta_b \cdot \overline{SAT}(\text{prep}_1(G)) \geq \beta_b \cdot (\beta_a \cdot \overline{SAT}(G))$$

  By setting $\beta_1 = \beta_a \cdot \beta_b$, which is a positive constant, we have proven the lemma.

6.2 Proof of Gap Amplification

Original Lemma:

Lemma 6.4 (Amplification Lemma)**.** *Let G be a d-regular constraint graph with self-loops and $\lambda_2(G) \leq \lambda < d$. There exists a constant $\beta_2 > 0$ (depending on $\lambda, d, |\Sigma|$) such that for any $t \in \mathbb{N}$, the powered graph G^t satisfies:*

- *(Completeness) If $\overline{SAT}(G) = 0$, then $\overline{SAT}(G^t) = 0$.*
- *(Soundness) $\overline{SAT}(G^t) \geq \beta_2\sqrt{t} \cdot \min\left(\overline{SAT}(G), \frac{1}{t}\right)$.*

This is the main technical lemma of the proof. It shows that the "Powering" operation successfully amplifies the gap $\overline{SAT}(G)$.

Proof. We will prove the Completeness and Soundness properties separately.

1. Completeness We must show that if $\overline{SAT}(G) = 0$, then $\overline{SAT}(G^t) = 0$.

1. If $\overline{SAT}(G) = 0$, there exists a "perfect" assignment $\sigma : V \to \Sigma$ that satisfies every constraint in G.
2. We construct a "truthful" assignment $\vec{\sigma} : V \to \Sigma'$ for the new graph G^t. An assignment $\vec{\sigma}(u)$ for a vertex u is its "opinion" of all vertices $w \in \Gamma(u)$ (its $\lceil t/2 \rceil$-step neighborhood). We define this truthful opinion to be the *actual* value of w under the perfect assignment σ. That is:
$$\forall u \in V, \forall w \in \Gamma(u), \quad \vec{\sigma}(u)_w = \sigma(w)$$
3. Now, consider any constraint $c^{e'}$ on an edge $e' = (u, v)$ in G^t. By definition, this constraint is satisfied if there exists a local assignment σ_{local} that:
   - (a) is consistent with $\vec{\sigma}(u)$'s opinion;
   - (b) is consistent with $\vec{\sigma}(v)$'s opinion;
   - (c) satisfies all original constraints from G between $\Gamma(u)$ and $\Gamma(v)$.
4. The global perfect assignment σ is this σ_{local}!
   - (a) σ is consistent with $\vec{\sigma}(u)$ (by our definition of $\vec{\sigma}$).
   - (b) σ is consistent with $\vec{\sigma}(v)$ (by our definition of $\vec{\sigma}$).
   - (c) σ satisfies all constraints in G (by our initial premise).

Since we have constructed a perfect assignment $\vec{\sigma}$ for G^t, it follows that $\overline{SAT}(G^t) = 0$.

2. Soundness This is the core of the proof. We are given an optimal assignment $\vec{\sigma}$ for G^t such that $\overline{SAT}(G^t) = \overline{SAT}_{\vec{\sigma}}(G^t)$. We must show this gap is large if $\overline{SAT}(G)$ is non-zero.

Setup Let $\vec{\sigma}$ be the optimal assignment for G^t, so $\overline{SAT}(G^t) = \overline{SAT}_{\vec{\sigma}}(G^t)$. First, we "decode" the G^t assignment $\vec{\sigma}$ into a "popular opinion" assignment σ for G. For each vertex $v \in V$, its value $\sigma(v)$ is defined as the value $a \in \Sigma$ that is most popular in the "opinions" held by its $\lceil t/2 \rceil$-step neighbors about v :
$$\sigma(v) \triangleq \operatorname*{argmax}_{a \in \Sigma} \left(\Pr_{w \in \Gamma(v)} [\vec{\sigma}(w)_v = a] \right)$$
By definition of σ, the probability of this popular opinion is at least $1/|\Sigma|$ within the neighbourhood. Let F_σ be the set of all edges in G that this decoded assignment σ fails to satisfy, so $\overline{SAT}_\sigma(G) = |F_\sigma|/|E|$. By definition of the minimum gap, $\overline{SAT}(G) \leq \overline{SAT}_\sigma(G)$.

Now, we define F as an arbitrary **subset** of F_σ whose size is **capped**:
$$\frac{|F|}{|E|} \triangleq \min \left(\overline{SAT}_\sigma(G), \frac{1}{t} \right)$$
Our goal is to show that $\overline{SAT}(G^t) \geq \Omega(\sqrt{t}) \cdot \frac{|F|}{|E|}$, which will imply the lemma.

Let $I = \{i \in \mathbb{N} \mid \frac{t}{2} - \sqrt{t/2} < i \leq \frac{t}{2} + \sqrt{t/2}\}$ be the set of "middle" indices. $|I| = \Omega(\sqrt{t})$.

We define a random variable $N(e')$ over the space of all t-step walks $e' \in E'$. This variable will count the number of "hits" in the "middle" portion of the walk.

Definition 6.5 (Hit). *Let $I = \{i \in \mathbb{N} \mid \frac{t}{2} - \sqrt{t/2} < i \leq \frac{t}{2} + \sqrt{t/2}\}$ be the set of "middle" indices. A t-walk $e' = (v_0, \ldots, v_t)$ is **hit** by its i-th edge (v_{i-1}, v_i) if both of the following conditions hold:*

1. ***Edge Violation:*** *The edge in G is violated by σ, i.e., $(v_{i-1}, v_i) \in F$.*
2. ***Opinion Agreement:*** *The "opinions" of the walk's endpoints $\vec{\sigma}(v_0)$ and $\vec{\sigma}(v_t)$ agree with the popular opinion σ at the point of the hit:*

$$\vec{\sigma}(v_0)_{v_{i-1}} = \sigma(v_{i-1}) \quad \textit{and} \quad \vec{\sigma}(v_t)_{v_i} = \sigma(v_i)$$

Let $N(e') = |\{i \in I \mid e' \text{ is hit by its } i\text{-th edge}\}|$ be the total number of middle hits for a walk e'.

Hits Imply Rejection If $N(e') > 0$, the walk e' is hit at some step $i \in I$. By condition (1), the assignment $(\sigma(v_{i-1}), \sigma(v_i))$ violates the G-constraint $c^{(v_{i-1}, v_i)}$. By condition (2), the G^t-assignment $(\vec{\sigma}(v_0), \vec{\sigma}(v_t))$ is being asked to be consistent with this violating assignment. Therefore, if $N(e') > 0$, the constraint $c^{e'}(\vec{\sigma}(v_0), \vec{\sigma}(v_t))$ must be **FALSE**.

$$\overline{SAT}(G^t) = \Pr_{e' \in E'}[e' \text{ rejects } \vec{\sigma}] \geq \Pr_{e' \in E'}[N(e') > 0]$$

We can bound this probability using the second moment method:

$$\Pr[N(e') > 0] \geq \frac{(\mathbb{E}[N(e')])^2}{\mathbb{E}[N(e')^2]}$$

The proof is reduced to finding a lower bound on $\mathbb{E}[N]$ and an upper bound on $\mathbb{E}[N^2]$.

Lemma 6.6 (First Moment Bound). *The expected number of middle hits is:*

$$\mathbb{E}_{e'}[N(e')] \geq \Omega(\sqrt{t}) \cdot \frac{|F|}{|E|}$$

Proof. Let N_i be the indicator variable that a walk e' is hit at step i. By linearity of expectation, $\mathbb{E}[N] = \sum_{i \in I} \mathbb{E}[N_i] = \sum_{i \in I} \Pr[N_i = 1]$. A random walk $e' = (v_0, \ldots, v_t)$ can be chosen by: (1) picking a random edge $(u, v) = (v_{i-1}, v_i) \in E$, (2) taking a random $(i-1)$-walk "backwards" from u to v_0, and (3) taking a random $(t-i)$-walk "forwards" from v to v_t. The probability of a hit at i is the product of the probabilities of the three independent events (from the definition of a hit):

$$\Pr[N_i = 1] = \Pr[(u, v) \in F] \cdot \Pr[\vec{\sigma}(v_0)_u = \sigma(u)] \cdot \Pr[\vec{\sigma}(v_t)_v = \sigma(v)]$$

The first term is simply $\frac{|F|}{|E|}$. For the second term, $p_u = \Pr[\vec{\sigma}(v_0)_u = \sigma(u)]$, the walk has length $i-1$. Since $i \in I$, $i - 1 \approx t/2$. Because G is an expander with self-loops, the distribution of a random $(i-1)$-walk is very close to that of a random $(t/2)$-walk (This is argued more precisely in the paper, I am leaving that out for the sake of brevity.). By definition, $\sigma(u)$ is the *most popular* value for a $(t/2)$-walk, so its probability is at least $1/|\Sigma|$. Thus, $p_u \geq \Omega(1/|\Sigma|) = \Omega(1)$. The same logic applies to $p_v = \Pr[\vec{\sigma}(v_t)_v = \sigma(v)]$, since its walk length $t - i$ is also $\approx t/2$. Therefore, $\Pr[N_i = 1] \geq \frac{|F|}{|E|} \cdot \Omega(1) \cdot \Omega(1) = \Omega(|F|/|E|)$. Summing over all $|I| = \Omega(\sqrt{t})$ indices in the middle:

$$\mathbb{E}[N] = \sum_{i \in I} \Pr[N_i = 1] \geq |I| \cdot \Omega(|F|/|E|) \geq \Omega(\sqrt{t}) \cdot \frac{|F|}{|E|}$$

□

Lemma 6.7 (Second Moment Bound)**.** *To control over-counting, we bound the second moment.*

$$\mathbb{E}_{e'}[(N(e'))^2] \leq O(\sqrt{t}) \cdot \min\left(\overline{SAT}_\sigma(G), \frac{1}{t}\right)$$

Proof. We define a simpler random variable $Z(e') = |\{i \in I \mid e'_i \in F\}|$, which just counts the number of times the walk *passes through* F in the middle. Since $N(e')$ only counts hits (which requires $e'_i \in F$), we have $N(e') \leq Z(e')$ for all e'. Thus, $\mathbb{E}[N^2] \leq \mathbb{E}[Z^2]$. Let Z_i be the indicator that $e'_i \in F$. Then $Z = \sum_{i \in I} Z_i$.

$$\mathbb{E}[Z^2] = \mathbb{E}\left[\left(\sum_{i \in I} Z_i\right)^2\right] = \sum_{i \in I} \mathbb{E}[Z_i^2] + \sum_{i \neq j \in I} \mathbb{E}[Z_i Z_j]$$

Diagonal terms: $\mathbb{E}[Z_i^2] = \mathbb{E}[Z_i] = \Pr[e'_i \in F] = \frac{|F|}{|E|}$.

$$\sum_{i \in I} \mathbb{E}[Z_i^2] = |I| \cdot \frac{|F|}{|E|} = \Omega(\sqrt{t}) \cdot \frac{|F|}{|E|}$$

Off-diagonal terms: Assume $i < j$. $\mathbb{E}[Z_i Z_j] = \Pr[Z_i = 1 \wedge Z_j = 1] = \Pr[Z_i = 1] \cdot \Pr[Z_j = 1 \mid Z_i = 1]$.

$$\Pr[Z_j = 1 \mid Z_i = 1] = \Pr[e'_j \in F \mid e'_i \in F]$$

This is the probability that a random walk that just traversed an edge in F will, $j - i$ steps later, traverse another edge in F. We use the **Random Walk Estimate (Theorem 3.8)**:

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

So, $\mathbb{E}[Z_i Z_j] \leq \frac{|F|}{|E|}\left(\frac{|F|}{|E|} + (\frac{\lambda}{d})^{j-i-1}\right)$. Now we sum over all $O(t)$ pairs (i, j) with $i \neq j$:

$$\sum_{i \neq j \in I} \mathbb{E}[Z_i Z_j] \leq \sum_{i \neq j \in I} \left(\frac{|F|}{|E|}\right)^2 + \sum_{i \neq j \in I} \frac{|F|}{|E|} \left(\frac{\lambda}{d}\right)^{|j-i|-1}$$

We use our assumption $\frac{|F|}{|E|} \leq \min(\overline{SAT}_\sigma(G), 1/t) \leq 1/t$. Term 1: $\sum_{i \neq j \in I} (\frac{|F|}{|E|})^2 \leq |I|^2 \cdot (1/t)^2 = O(t) \cdot (1/t^2) = O(1/t)$. Term 2: The inner sum $\sum_{j \neq i} (\lambda/d)^{|j-i|-1}$ is a geometric series that we can bound by a constant $C_{\lambda,d}$. So this term is $\sum_{i \in I} \frac{|F|}{|E|} \cdot C_{\lambda,d} = |I| \cdot \frac{|F|}{|E|} \cdot C_{\lambda,d} = O(\sqrt{t}) \cdot \frac{|F|}{|E|}$. Combining the diagonal and off-diagonal terms:

$$\mathbb{E}[N^2] \leq \mathbb{E}[Z^2] \leq \left(\Omega(\sqrt{t})\frac{|F|}{|E|}\right) + \left(O(\sqrt{t})\frac{|F|}{|E|} + O(1/t)\right)$$

The dominant term is $O(\sqrt{t})\frac{|F|}{|E|}$, so $\mathbb{E}[N^2] \leq O(\sqrt{t}) \cdot \frac{|F|}{|E|}$. By our choice of F, this is $O(\sqrt{t}) \cdot \min\left(\overline{SAT}_\sigma(G), \frac{1}{t}\right)$. □

Conclusion We now substitute the moment bounds back into the inequality:

$$\overline{SAT}(G^t) \geq \Pr[N(e') > 0] \geq \frac{(\mathbb{E}[N(e')])^2}{\mathbb{E}[N(e')^2]}$$

$$\overline{SAT}(G^t) \geq \frac{\left(\Omega(\sqrt{t}) \cdot \frac{|F|}{|E|}\right)^2}{O(\sqrt{t}) \cdot \frac{|F|}{|E|}}$$

$$\overline{SAT}(G^t) \geq \frac{\Omega(t) \cdot (|F|/|E|)^2}{O(\sqrt{t}) \cdot |F|/|E|}$$

$$\overline{SAT}(G^t) \geq \Omega(\sqrt{t}) \cdot \frac{|F|}{|E|}$$

Let β_2 be the positive constant hidden in $\Omega()$.

$$\overline{SAT}(G^t) \geq \beta_2 \sqrt{t} \cdot \frac{|F|}{|E|}$$

Finally, we substitute back the definition of $|F|/|E|$:

$$\overline{SAT}(G^t) \geq \beta_2 \sqrt{t} \cdot \min\left(\overline{SAT}_\sigma(G), \frac{1}{t}\right)$$

Since $\overline{SAT}(G) \leq \overline{SAT}_\sigma(G)$, it is also true that $\min\left(\overline{SAT}(G), \frac{1}{t}\right) \leq \min\left(\overline{SAT}_\sigma(G), \frac{1}{t}\right)$. This gives the final result:

$$\overline{SAT}(G^t) \geq \beta_2 \sqrt{t} \cdot \min\left(\overline{SAT}(G), \frac{1}{t}\right)$$

This completes the proof of soundness. □

6.3 Proof of Composition

Original Lemma:

Lemma 6.8 (Composition Lemma)**.** *Assume the existence of an assignment tester $\mathcal{P}$ with a constant rejection probability $\epsilon > 0$ and a constant-size alphabet Σ_0. There exist constants $\beta_3 > 0$ (depending on $\mathcal{P}$) and $C > 0$ (depending on $\mathcal{P}$ and $|\Sigma|$) such that the following holds.*

Given any constraint graph $G = \langle (V, E), \Sigma, \mathcal{C} \rangle$, one can compute, in time linear in $size(G)$, the constraint graph $G' = G \circ \mathcal{P}$, such that:

- $size(G') \leq C \cdot size(G)$
- $\beta_3 \cdot \overline{SAT}(G) \leq \overline{SAT}(G') \leq \overline{SAT}(G)$

Proof. We must prove the three properties claimed by the lemma: linear-time/size, completeness, and soundness.

1. Linear Time and Size The construction of $G' = G \circ \mathcal{P}$ involves two steps for each of the $|E|$ edges of G:

1. **Robustization:** We create a circuit Φ_e for each edge $e \in E$. The input to this circuit is $[u] \cup [v]$, which has $2l$ Boolean variables. The size of the alphabet (Σ) is a large constant, so $l = O(\log |\Sigma|)$ is also a constant. A circuit on a constant number of variables, $2l$, can be constructed in constant time and has a constant size, $O(2^{2l})$.
2. **Composition:** We run the assignment tester $\mathcal{P}$ on the constant-sized circuit Φ_e. Since the *input* to $\mathcal{P}$ is of constant size, the *output* graph $G_e = \mathcal{P}(\Phi_e)$ also has a constant size, which we can call C_e.

The final graph G' is the union of $|E|$ of these constant-sized gadgets. The total number of new vertices and edges is $O(|E| \cdot C_e)$. Since G is a constant-degree graph (from Preprocessing), $|V| = O(|E|)$, so $size(G) = |V| + |E| = O(|E|)$. Therefore, $size(G') = O(|E|) = O(size(G))$. This entire construction (where we run $|E|$ constant-time operations) takes $O(|E|)$ time, which is linear in $size(G)$. We can set C to be the constant $O(C_e)$.

2. Completeness We must show that $\overline{SAT}(G) = 0 \implies \overline{SAT}(G') = 0$.

- Assume $\overline{SAT}(G) = 0$. This means there exists a "perfect" assignment $\sigma : V \to \Sigma$ that satisfies every constraint $c^e \in \mathcal{C}$.
- We construct an assignment σ' for G' as follows:
  1. For the "cluster" variables $[v]$ for each $v \in V$, we set their assignment to be the correct encoding of $\sigma(v)$:
  $$\sigma'([v]) = \text{Enc}(\sigma(v))$$
  2. Now, consider any gadget $G_e = \mathcal{P}(\Phi_e)$. Since σ satisfies c^e, the assignment $\sigma'([u]) \cup \sigma'([v])$ is, by definition, a satisfying assignment for the circuit Φ_e.
  3. By the **Completeness of the Assignment Tester** $\mathcal{P}$, since $\sigma'|_{[u]\cup[v]}$ is in $SAT(\Phi_e)$, there exists an extension b for the helper variables in G_e (i.e., $V_e \setminus ([u] \cup [v])$) such that $\overline{SAT}_{\sigma' \cup b}(G_e) = 0$.
  4. We use this b to complete our assignment σ'.
- Since this holds for all gadgets G_e, the assignment σ' satisfies every constraint in the final union G'. Thus, $\overline{SAT}(G') = 0$.

By the same construction we can show that $\overline{SAT}(G') \leq \overline{SAT}(G)$.

3. Soundness We must show that $\overline{SAT}(G') \geq \beta_3 \cdot \overline{SAT}(G)$.

- Let σ' be the optimal assignment for G', such that $\overline{SAT}_{\sigma'}(G') = \overline{SAT}(G')$.
- For each $v \in V$, we look at the l-bit string $\sigma'([v])$ and find the symbol $s \in \Sigma$ whose encoding is closest to it:
  $$\sigma(v) = \arg\min_{s \in \Sigma} \{\text{dist}(\sigma'([v]), \text{Enc}(s))\}$$
  (Ties are broken arbitrarily).
- Let $F \subseteq E$ be the set of edges in G that are violated by this decoded assignment σ. By definition of $\overline{SAT}(G)$, we have $\overline{SAT}_{\sigma}(G) \geq \overline{SAT}(G)$.
- The total gap in G' is the average of the gaps in its gadgets. We can lower-bound this by only summing over the gadgets corresponding to the violated edges F:
  $$\overline{SAT}(G') = \overline{SAT}_{\sigma'}(G') = \frac{1}{|E|} \sum_{e \in E} \overline{SAT}_{\sigma'}(G_e) \geq \frac{1}{|E|} \sum_{e \in F} \overline{SAT}_{\sigma'}(G_e)$$
  (This assumes all gadgets G_e have the same size, which we can assume w.l.o.g.).
- Now, we just need a lower bound for $\overline{SAT}_{\sigma'}(G_e)$ for an edge $e \in F$.
- Let $e = (u, v) \in F$. Let $a = \sigma'|_{[u]\cup[v]}$ be the (Boolean) assignment σ' gives to the inputs of the circuit Φ_e.
- By the **Soundness of the Assignment Tester** $\mathcal{P}$, we know:
  $$\overline{SAT}_{\sigma'}(G_e) \geq \epsilon \cdot \text{rdist}(a, SAT(\Phi_e))$$
- We need to lower-bound $\text{rdist}(a, SAT(\Phi_e))$. Let $\rho > 0$ be the relative distance of our encoding Enc.
- An assignment is in $SAT(\Phi_e)$ if it's a valid encoding $(\text{Enc}(a_u), \text{Enc}(a_v))$ of a pair (a_u, a_v) that satisfies c^e.
- The assignment a is $(\sigma'([u]), \sigma'([v]))$. The decoded assignment is $(\sigma(u), \sigma(v))$. Since $e \in F$, this decoded pair *violates* c^e.

- By our "closest-distance" decoding rule, $\sigma'([u])$ is closer to $\text{Enc}(\sigma(u))$ than to any other $\text{Enc}(a_u)$. This implies that $\text{rdist}(\sigma'([u]), \text{Enc}(a_u)) \geq \rho/2$ for any $a_u \neq \sigma(u)$.
- Any "good" assignment in $SAT(\Phi_e)$ must be an encoding of a *different* pair (a_u, a_v). Therefore, it must be at least $\rho/2$ far from a in *either* the $[u]$ part or the $[v]$ part. This means the relative distance (over the whole $2l$-bit string) is at least $\rho/4$.

$$\text{rdist}(a, SAT(\Phi_e)) \geq \rho/4$$

- Plugging this back in: $\overline{SAT}_{\sigma'}(G_e) \geq \epsilon \cdot (\rho/4)$.
- Let $\beta_3 = \epsilon \cdot \rho/4$. Since ϵ and ρ are positive constants, β_3 is a positive constant.
- We can now finish the main soundness argument:

$$\overline{SAT}(G') \geq \frac{1}{|E|} \sum_{e \in F} \overline{SAT}_{\sigma'}(G_e) \geq \frac{1}{|E|} \sum_{e \in F} \beta_3 = \frac{|F| \cdot \beta_3}{|E|} = \beta_3 \cdot \frac{|F|}{|E|} = \beta_3 \cdot \overline{SAT}_{\sigma}(G)$$

- Since $\overline{SAT}_{\sigma}(G) \geq \overline{SAT}(G)$, we have $\overline{SAT}(G') \geq \beta_3 \cdot \overline{SAT}(G)$.

This proves all parts of the lemma. □

6.4 Final Thoughts and Impact

The PCP Theorem is one of the cornerstone results in computational complexity. It asserts that any NP-witness can be encoded in a way that allows for efficient probabilistic verification by reading only a constant number of bits. The result, suprisingly, is also equivalent to the hardness of approximating many critical optimization problems. Irit Dinur's proof that I primarily focused on, is an elegant and combinatorial approach unlike the original algebraic proofs. It's central idea of gap amplification to repeatedly increasing the $\overline{SAT}$ value of contsraint graphs and thereby building the gap between satisfiable and unsatisfiable instances is both intuitive and powerful.

References

[1] Min Jae Song, "An Introduction to the PCP Theorem".

[2] Irit Dinur, "The PCP Theorem by Gap Amplification", 2005 draft, 2007 official.

[3] Sanjeev Arora and Boaz Barak, "Computational Complexity: A Modern Approach"