W. Remmerswaal

Quantum advantage for combinatorial optimization problems

Bachelor thesis

2026

Thesis supervisors: prof.dr. V. Dunjko
prof.dr. S.O. Fehr

Leiden University
Mathematical Institute
Leiden Institute of Advanced Computer Science

Abstract

Quantum computers are known to outperform classical computers on certain problems, most notably on integer factoring due to Shor's algorithm [20]. This thesis studies whether a similar quantum advantage exists for approximating optimization problems. We present a generalization and a detailed proof of the statement made by Szegedy [22] that, assuming that the problem of integer factoring is not solvable in polynomial time, there exists an optimization problem for which there exists a polynomial-time quantum 0.99-approximation algorithm but no polynomial-time classical $(\frac{7}{8} + 0.01)$-approximation algorithm. The proof combines the Cook–Levin theorem, the PCP theorem, and Håstads optimal inapproximability result for MAX-3SAT. Finally, we show that the statement still holds when replacing the assumption that integer factoring is not solvable in polynomial time with the assumption that there exists a problem that is not in **P**, is in **NP** and admits a polynomial-time quantum algorithm finding an **NP** witness.

Contents

1 Introduction

In the field of computational complexity we study the resources required for computing the solution to computational problems. Quantum computing introduces a different model of computation that comes with an improvement on the amount of resources required to solve certain problems, compared to their classical variant. The central question in this thesis was raised by Eisert et al. [17], namely whether a quantum improvement can be found on the run time of algorithms solving combinatorial optimization problems – that is, problems of optimizing a target function over a finite set of solutions. Eisert et al. and Szegedy [22] both give a proof confirming such an improvement for different problems. In this thesis, we work out the details of the statement by Szegedy.

Computational complexity theory is concerned with solving different classes of problems. Decision problem are the problem of answering a yes or no question. For example, is an integer prime or not. Search problems generalize decision problems. They require us to find a specific solution to a problem instance. For example, finding a prime factor of some positive integer is a search problem. A specific type of search problem is the optimization problem, where we distinguish between maximization and minimization problems. The goal is to optimize a target function over a search space. For example, consider a system of equations where we want to maximize the number of equations that are satisfied at once.

To formalize the concept of an algorithm and to analyze algorithms solving these problems, we require a model of computation. In 1936, Alan Turing [23] introduced the Turing machine, a model of computation whose strength lies in its ability to simulate most other models of computation with only a polynomial loss in running time.

An important resource for a Turing machine is the run-time. The time complexity of a Turing machine is given by the function of the input size of a problem instance that counts the number of computational steps of the Turing machine on an input of this size. Different inputs of the same size may cause different run-times of a Turing machine. Throughout this thesis, we are only interested in the maximum running time of a Turing machine over all inputs of the same size. We call this the worst-case time complexity, or in the scope of this thesis just the time complexity of a Turing machine.

In computational complexity, we study the asymptotic properties of this function. A polynomial-time Turing machine is a Turing machine that has time complexity $O(p(n))$, for some polynomial p. Polynomial time-complexity is often regarded as a practical[1] notion of efficient computation.

Given a model of computation and a notion of the time complexity of an algorithm, we can define the class of problems that are "efficiently" solvable. The complexity class **P** contains all decision problems that can be solved by a polynomial-time Turing machine.

Another important class of decision problems is **NP**. A decision problem is in **NP** if there exists a polynomial-time Turing machine, called a verifier, with the following properties. The verifier takes as input an instance and a binary string, called a *witness*.

- For the "yes-instances", there exists a witness that causes the verifier to accept.
- For "no-instances", there exists no such witness.

A witness can be seen as a mathematical proof of the instance being a "yes-instance". The verifier checks the validity of this proof.

This leads to a fundamental question in complexity theory: if the existence of a solution can be verified efficiently, can the existence of such a solution be decided efficiently? In complexity theoretic terms, this question is captured in the question whether the complexity classes **P** and **NP** are the same. It is widely

[1] On the condition that the degree and the constant factor of the polynomial are not too large: the time-complexity defined by the polynomials n^{100} or $10^{10^{10}}n$ is not feasible in any practical situation.

believed that they are not the same i.e. $\mathbf{P} \neq \mathbf{NP}$. Many complexity theoretic statements rely on the assumption of this statement. Intuitively, the ability to efficiently verify a proof does not seem sufficient to efficiently find such a proof. Despite decades of research, there is no proof that shows whether **P** and **NP** are equal. This question remains a central open question in the field of complexity theory and is included in the list of Millenium Problems of the Clay Mathematics Institute. [7].

In computational complexity theory, we also consider different models of computation, such as quantum computation. Recent research [15] has shown that quantum computation provides a significant speed-up on certain problems, compared to its classical alternative. However, the precise relation between many classical and quantum complexity classes, such as the respective classes of decision problems solvable in polynomial time, is still unknown. There is no problem that is known to have a proven super-polynomial quantum advantage; that is, there is no known problem that is proven to be solvable in polynomial time by a quantum computer but not be solvable in polynomial time by any classical computer.

The problem of finding a prime factor of an integer provides a strong piece of evidence for a super-polynomial quantum speed-up. Classically, it is widely believed, but not proven, that no polynomial-time algorithm exists that solves the factoring problem. Shor's algorithm [20] shows us that it can be solved in polynomial time on a quantum computer.

The focus of this thesis is on optimization problems. To decrease the difficulty of solving this type of problem, one can relax the requirement of computing an optimal solution to that of computing a solution that is close to optimal. Some optimization problems can be approximated arbitrarily close to their optimum. That is, for each $\epsilon > 0$, there exists a Turing machine that given an instance, outputs a valid solution with a value that is at least a $1 - \epsilon$ fraction of the optimal value of this instance. Others, such as the MAX-3SAT problem, which we introduce later, can not be approximated arbitrarily closely, unless $\mathbf{P} = \mathbf{NP}$.

Eisert et al. [17] raise the question whether there is a super-polynomial quantum speed-up possible for the approximation of combinatorial optimization problems, that is, optimization problems whose search space is finite. They answer this question in the affirmative, on the standard assumption the RSA function is classically hard to invert [19]. Specifically, they construct a combinatorial optimization problem for which a polynomial-time quantum Turing machine achieves a strictly higher factor of approximation than any classical polynomial-time Turing machine. The instances of this maximization problem are artificially created and highly structured. It is unlikely that these will arise naturally. However, the proof of this theorem still gives us insight in how quantum computers might provide an improvement for approximating optimization problems [17].

A similar result follows from known complexity theoretic results in the field of PCP theory. M. Szegedy [22] states that, on the assumption that the problem of finding prime factors of an integer is classically hard, there exists an optimization problem such that it is classically hard to approximate it with a factor $\frac{7}{8} + 0.01$, while we can approximate it with a factor 0.99 using a quantum Turing machine. While these instances are also highly structured, Eisert et al. remark that it would be beneficial to explicitly work out the details of Szegedy's argument. In this thesis, we lay out these details and propose two generalizations. Namely, a generalization of the assumption that integer factoring is hard and a generalization of the approximation factor.

In section 2, we introduce the mathematical basic for computational complexity theory. We state two claims made by Szegedy and prove the main theorem based on these claims in section 3. The main theorem is a version of the statement made by Szegedy, but with generalized approximation factors. In section 4, we present the PCP theory required for these claims. These steps are all known results from PCP theory. We finish the proof of the main theorem in section 5 by proving the two claims. In chapter 6, we propose an expansion of main theorem. This expansion broadens the assumption that the factoring problem is hard classically and efficient on a quantum Turing machine.

2 Introduction to computational complexity

In this section we lay out the mathematical foundation of computational complexity theory. We define formalizations of different classes of computational problems and introduce important examples of these problems. We define the Karp reduction, which forms the basis of **NP**-complete problems. Solving these problems in polynomial time means solving all **NP** problems in polynomial time. Next, we introduce promise problems and optimization problems and define the concept of an approximation algorithm. Lastly, we introduce quantum computation and various quantum computational classes.

2.1 Decision problems and complexity classes

2.1.1 Languages

Let Σ be an arbitrary finite set that will be fixed in this section. We call this set an alphabet. We denote

$$\Sigma^* := \bigcup_{n \in \mathbb{N}_0} \Sigma^n$$

for the set of finite sequences, which we call strings. We use the symbol ε for the empty string and by convention $\Sigma^0 = \{\varepsilon\}$. We define the size of $x \in \Sigma^*$ to be

$$|x| := n$$

where $n \in \mathbb{N}_0$ is the unique integer such that $x \in \Sigma^n$.

A language is a set of strings over some alphabet.

Definition 2.1. A *language* over some alphabet Σ is a subset $L \subset \Sigma^*$.

Languages capture the concept of a *decision problem*. We associate the language L with the problem of deciding whether some $x \in \Sigma^*$ is in the language L or not.

In computational complexity theory, we are interested in the amount of resources required to solve these problems computationally. To formalize this, we use Turing machines as a model of computation. We do not go into detail on the inner workings of a Turing machine and refer the reader to Sipser [21, Chapter 3] for a formal definition. For this thesis it is sufficient to think of a Turing machine as a formal model of an algorithm.

We say that a Turing machine runs in polynomial time if there exists some polynomial $p : \mathbb{N} \to \mathbb{N}$ such that on any input $x \in \Sigma^*$, the Turing machine halts after at most $p(|x|)$ computational steps. A Turing machine running in polynomial time with respect to the input size is generally considered efficient and scalable.

Definition 2.2. The complexity class P consists of all languages that are decided by a polynomial-time Turing machine.

As explained in [21, Chapter 3], we do not describe a polynomial-time Turing machine on a low level as this can be quite cumbersome and distracts us from the purpose of the machine. Instead, we describe a high level algorithm and argue that it can be implemented by a polynomial-time Turing machine. Many base operations, such as adding two numbers or testing two variables for equality, can be simulated by a polynomial-time Turing machine. Since the composition of a polynomial number of polynomial-time operations still runs in polynomial time, we often treat these types of base operations as constant time operations.

2.1.2 Encodings

Some decision problems are not naturally given in the form of a language. We represent these problems as a language by using a suitable encoding. For example, the set of natural numbers $\mathbb{N}$ is not a finite set and thus not an alphabet. We can still encode a positive integer with its binary representation.

The precise language representing a decision problem depends on the choice of encoding. So, we only use encodings that a Turing machine can switch between in polynomial time. For this reason we will not specify the encoding for a problem and, by abuse of notation, we identify an object with its encoding.

Any alphabet can itself be encoded by the binary alphabet $\{0,1\}$. Since any finite alphabet can be enumerated, we can represent a symbol with the binary representation of its index. For this reason, unless specified otherwise, we take $\{0,1\}$ as our default alphabet.

When solving decision problems, view any input $x \in \{0,1\}^*$ as the binary encoding of a problem instance. If a binary string $x \in \{0,1\}^*$ is not the valid encoding of some problem instance, we assume that the Turing machine solving the problem immediately rejects it.

2.1.3 Boolean formulas

When working with binary values, we identify 1 with *true* and 0 with *false*. A boolean formula over n binary variables $(x_1 \dots x_n)$ is an expression built recursively from variables, negations, conjunctions and disjunctions. We use the over bar $\overline{\ }$ to denote negation. For example, $x_1 \vee \overline{(x_2 \wedge \overline{x_3})}$ is a boolean formula.

A boolean formula ϕ over n variables naturally defines a function $f_\phi : \{0,1\}^n \to \{0,1\}$, where the output is the expression evaluated on the input binary string. By abuse of notation, we identify a boolean formula with the function f_ϕ. We call a binary string $a \in \{0,1\}^n$ an *assignment* for ϕ and say that *a satisfies* ϕ if $\phi(a) = 1$.

We denote Φ_n for the set of all boolean formulas consisting of exactly n variables. An important decision problem involving boolean formulas asks whether a boolean formula ϕ is satisfiable. In other words, does there exist a satisfying assignment for ϕ? This problem is captured by the following language.

Definition 2.3.

$$\mathsf{SAT} := \bigcup_{n \in \mathbb{N}} \{\phi \in \Phi_n \big| \exists a \in \{0,1\}^n, \phi(a) = 1\}$$

Note that the language should actually be defined as the set of encodings of satisfiable boolean formulas. However, as discussed before, we omit specifying the encoding.

In many cases we work with a structured subset of boolean formulas. We call a variable x or its negation $\overline{x}$ a *literal* and we say that a boolean formula is in k *conjunctive normal form (kCNF)*, if the formula is a conjunction of *clauses*, where each clause is the disjunction of exactly k literals over distinct variables. For example, the boolean formula

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_2} \vee \overline{x_4} \vee x_5)$$

is a 3CNF formula. We define a language that captures the decision problem of deciding whether a 3CNF formula is satisfiable. Let $\Phi_n^{k\text{CNF}} \subset \Phi_n$ be the set of all kCNF boolean formulas consisting of exactly n.

Definition 2.4.

$$\mathsf{kSAT} := \bigcup_{n \in \mathbb{N}} \{\phi \in \Phi_n^{k\text{CNF}} \big| \exists a \in \{0,1\}^n, \phi(a) = 1\}$$

The notation EkSAT is commonly used for the problem of the satisfiability of kCNF formulas with each clause containing exactly k literals over distinct variables. In this thesis however, we only work with kCNF formulas of this form, so we use the kSAT notation.

2.1.4 Karp reductions and NP

We say that a function $f : X \to Y$ is polynomial-time computable if there exists a polynomial-time Turing machine that given $x \in X$ outputs $f(x)$.

Definition 2.5. Let $L \subset \{0,1\}^*$ and $L' \subset \{0,1\}^*$ be two languages. A Karp reduction from L to L' is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$ it holds that

$$x \in L \Leftrightarrow f(x) \in L'$$

Let A be a Turing machine deciding the language L' in polynomial time. We can use a Karp reduction f from language L to L' to decide L in polynomial time. Given $x \in \{0,1\}^*$, compute $f(x)$ and run A on $f(x)$ to decide whether $f(x) \in L'$. Since $x \in L \Leftrightarrow f(x) \in L'$, this determines whether $x \in L$.

It is easy to verify that Karp reductions are closed under composition. We have already introduced the complexity class **P** consisting of languages that can be decided in polynomial time by a Turing machine. The complexity class **NP** consists of the languages that can be *verified* in polynomial time.

Definition 2.6. NP is the class of the languages where for each language L in **NP** the following holds. There exists a polynomial-time Turing machine V and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \{0,1\}^*$,

- if $x \in L$, then there exists some $w \in \{0,1\}^*$ with $|w| \leq p(|x|)$ such that $V(x,w) = 1$.
- If $x \notin L$, then for all $w \in \{0,1\}^*$ with $|w| \leq p(|x|)$ we have that $V(x,w) = 0$.

In other words, for any **NP** language L and any $x \in L$, there exists a proof of membership of bounded size, called a *witness*, that can be verified in polynomial time. For any $x \notin L$, there exists no such witness. A Turing machine that checks whether a proposed witness is correct is called a *verifier*.

Given a verifier V for an **NP** language L , we define the relation

$$R_L := \{(x,w) \big| V(x,w) = 1\}$$

By construction, it holds that

- $x \in L$ if and only if there exists a witness w such that $(x,w) \in R_L$.
- R_L is *polynomial-time verifiable*, meaning that membership in R_L can be decided in polynomial time.
- The size of a witness is bounded by a polynomial factor: there exists some polynomial p such that for each $(x,w) \in R_L$, we have that $|w| \leq p(|x|)$.

We will make use of this relation for languages in **NP** in the proof of the main theorem. Many (representations of) natural decision problems are in **NP**.

Lemma 2.1. *SAT* $\in$ ***NP*** *and 3SAT* $\in$ ***NP***

Proof. Let ϕ be a boolean formula over n variables. We define a verifier V that takes the input ϕ and $a \in \{0,1\}^*$. If $a \notin \{0,1\}^n$, V rejects and otherwise, V computes the value $\phi(a)$ and accepts if it is 1. To see that V runs in polynomial time, note that verifying $|a| = n$ and computing $\phi(a)$ both take polynomial time in $|\phi| + |a|$.

- If $\phi \in \mathsf{SAT}$, a satisfying assignment a causes V to accept.
- If $\phi \notin \mathsf{SAT}$, there is no satisfying assignment for ϕ, so no assignment causes V to accept.

The same proof is valid for 3CNF formulas. □

In the search of proving whether **P** = **NP**, we would like to find universal problems for **NP**, that is, problems such that if they can be solved in polynomial time, we can solve all **NP** problems in polynomial time.

A language L is called **NP**-hard under Karp reductions, or simply **NP**-hard, if for all languages $L' \in$ **NP**, there exists a Karp reduction from L' to L. A polynomial-time algorithm deciding an **NP**-hard language would imply that every language in **NP** can be decided in polynomial time. We say that a language is **NP**-complete if it is **NP**-hard and in **NP**. An important result in complexity theory shown independently by Cook [8] and Levin [14] is that indeed such universal languages exist.

Theorem 2.2 (Cook-Levin [8], [14])**.** *SAT is* ***NP****-complete.*

Proof. A proof of this theorem is given in section 4.1. □

2.2 Search problems and optimization problems

A broader class of problems than decision problems is the class of *search problems*. The task for these problems is to find a solution to the problem, rather than deciding whether one exists. We capture these problems with a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, where the first component represents problem instances and the second represents the solutions to these instances. As is the case with languages, a string $x \in \{0,1\}^*$ encodes an instance of the problem. Similarly, a string $y \in \{0,1\}^*$ encodes a candidate solution for an instance. Given an instance $x \in \{0,1\}^*$, we denote

$$S_x := \{y \in \{0,1\}^* \big| (x,y) \in R\}$$

for the set of *valid* or *feasible* solutions for x.

To effectively capture a search problem, we require the relation to be polynomial-time verifiable. The search problem associated with the polynomial-time verifiable relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is the computational task: given some $x \in \{0,1\}^*$, compute $y \in \{0,1\}^*$ such that $(x,y) \in R$.

Optimization problems are specified by the pair (R,m) containing a relation R and a target function m, where for any $x \in \{0,1\}^*$ and any feasible solution $y \in S_x$, $m(x,y) \in \mathbb{R}$ is the value of y for x that we want to optimize. For a maximization problem, the computational task is the following: given some problem instance $x \in \{0,1\}^*$, compute a feasible solution $y^* \in S_x$ such that

$$m(x,y^*) = \max_{y \in S_x} m(x,y)$$

Minimization problems are defined in a similar fashion, instead minimizing over the valid solutions instead of maximizing. We denote

$$\mathrm{OPT}_{(R,m)}(x) := m(x,y^*)$$

for the optimal value of x. We often write $\mathrm{OPT}(x)$ when the optimization problem is clear from the context.

An important optimization problem is the maximization version of 3SAT. Let

$$\phi := C_1 \wedge \ldots \wedge C_m$$

be a 3CNF formula over n variables, where each clause C_i is the disjunction of exactly three literals. For an assignment $a \in \{0,1\}^n$, we denote $N(\phi,a)$ for the number of clauses in ϕ that are satisfied by a. Let

$$R_{\mathrm{3CNF}} := \bigcup_{n \in \mathbb{N}} \{(\phi,a) \big| \phi \in \Phi_n^{\mathrm{3CNF}}, a \in \{0,1\}^n\} \tag{1}$$

define the relation of 3CNF formulas and any assignment to their variables.

The pair (R_{3CNF}, N) now defines the maximization problem MAX-3SAT: given some $\phi \in \Phi_n^{\mathrm{3CNF}}$, compute $a^* \in \{0,1\}^n$ such that

$$N(\phi,a^*) = \max_{a \in \{0,1\}^n} N(\phi,a)$$

2.2.1 Approximation algorithms

Finding the optimal solution to an optimization problem often requires a lot of resources. We relax the requirement of finding an optimal solution to finding an approximately optimal solution.

Definition 2.7. Let $0 < \alpha \leq 1$, let R be a relation and let m be a target function. An *α-approximation algorithm* for the maximization problem (R,m) is a Turing machine that given an instance $x \in \{0,1\}^*$ with $S_x \neq \emptyset$, outputs a valid solution $y \in S_x$ such that

$$m(x,y) \geq \alpha \, \mathrm{OPT}(x)$$

Approximation algorithms for minimization problems are defined analogously for any $\alpha \geq 1$. We can $\frac{7}{8}$ approximate MAX-3SAT in polynomial time.

Lemma 2.3. *There exists a polynomial-time $\frac{7}{8}$-approximation algorithm for MAX-3SAT.*

Proof. A random assignment satisfies a $\frac{7}{8}$ fraction of clauses in expectation. From this fact we can derive a deterministic algorithm that gives a $\frac{7}{8}$ approximation using the method of conditional expectation. We leave the details to the interested reader. □

A key result shown by Håstad [12] is that this factor of approximation is optimal under the assumption that $P \neq \mathbf{NP}$.

Theorem 2.4 ([12])**.** *For any $\epsilon > 0$, there exists no polynomial-time $(\frac{7}{8} + \epsilon)$-approximation algorithm for MAX-3SAT, unless $\boldsymbol{P = NP}$.*

Proof. The proof by Håstad will be shown in section 4.3. □

2.3 Promise problems

The final class of computational problems we introduce is the class of *promise problems*. A language L can be viewed as a partition of $\{0,1\}^*$ into two sets, L, the "yes-instances", and $\{0,1\}^* \setminus L$, the "no-instances". A promise problem partitions $\{0,1\}^*$ into three sets:

- L_{yes}, the "yes-instances"
- L_{no}, the "no-instances"
- $\{0,1\}^* \setminus (L_{yes} \cup L_{no})$, the "irrelevant" instances

The pair L_{yes}, L_{no} defines the promise problem of determining, given an input $x \in L_{yes} \cup L_{no}$, whether $x \in L_{yes}$ or $x \in L_{no}$. The set $L_{yes} \cup L_{no}$ is called the *promise*. An algorithm solving a promise problem is guaranteed to receive an input that lies in the promise. In the case of a promise problem, a string that is not the encoding of some instance simply falls in the category of irrelevant instances. In many cases we prefer working with promise problems instead of languages; see Goldreich [11] for further discussion.

We define many to one reductions from languages to promise problems and between promise problems.

Definition 2.8. Let L be a language, let $L' = (L'_{yes}, L'_{no})$ be a promise problem. A Karp reduction from L to L' is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for any $x \in \{0,1\}^*$ the following holds.

- If $x \in L$, then $f(x) \in L'_{yes}$.
- If $x \notin L$, then $f(x) \in L'_{no}$.

Definition 2.9. Let $L = (L_{yes}, L_{no})$ be a promise problem, let $L' = (L'_{yes}, L'_{no})$ be a promise problem. A Karp reduction from L to L' is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for any $x \in \{0,1\}^*$ the following holds.

- If $x \in L_{yes}$, then $f(x) \in L'_{yes}$.
- If $x \in L_{no}$, then $f(x) \in L'_{no}$.

2.3.1 Gap problems

We introduce terminology regarding the satisfiability of 3CNF formulas. Let ϕ be a 3CNF formula with m clauses and let $e \in [0,1]$ be a constant.

- We say that ϕ is *at most e-satisfiable* if $\mathrm{OPT}(\phi) \leq em$.
- We say that ϕ is *at least e-satisfiable* if $\mathrm{OPT}(\phi) \geq em$.

Many results on the hardness of approximation of optimizations problem rely on formulating the problem in terms of a promise problem. An important example is the problem of distinguishing between at least c-satisfiable 3CNF formulas and at most s-satisfiable 3CNF formulas, for some $0 \leq s < c \leq 1$.

Definition 2.10. Let $0 \leq s < c \leq 1$ be constants, let $\Phi_{3\text{CNF},c}$ be the set of at least c-satisfiable 3CNF formulas and let $\Phi_{3\text{CNF},s}$ be the set of at most s-satisfiable 3CNF formulas.

$$\mathsf{GAP}[c,s]\text{-}\mathsf{3SAT} = (\Phi_{3\text{CNF},c}, \Phi_{3\text{CNF},s})$$

An optimization problem naturally gives rise to an associated class of promise problems. Let (R, m) be an optimization problem and let $1 \geq c > s \geq 0$ be constants. Let $b : \{0,1\}^* \to \mathbb{R}$ be a polynomial-time computable upper bound on the optimal value, so $\mathrm{OPT}(x) \leq b(x)$ for all $x \in \{0,1\}^*$. Define

- $L_{yes} := \{x \in \{0,1\}^* \big| \, \mathrm{OPT}(x) \geq c \cdot b(x)\}$ and
- $L_{no} := \{x \in \{0,1\}^* \big| \, \mathrm{OPT}(x) \leq s \cdot b(x)\}$.

The promise problem associated with the optimization problem (R, m) is now given by the pair (L_{yes}, L_{no}) and we call this problem a *gap problem*. We call the set $L_{yes} \cup L_{no}$ a set of instances with a gap (c, s) and we say that this gap has size $c - s$.

In the context of gap problems, Karp reductions from a language to a promise problem and between promise problems are called gap-producing reductions and gap-preserving reductions respectively.

2.4 Quantum computation and integer factoring

Quantum complexity classes are defined using quantum Turing machines. In this thesis we are not interested in the low level details of such machines, so we do not provide a definition of a quantum Turing machine and refer the reader to Bernstein and Vazinari [6]. We define three important complexity classes arising from quantum Turing machines.

Definition 2.11. The complexity class **BQP** consists of languages L such that there exists a polynomial-time quantum Turing machine A such that for any input $x \in \{0,1\}^*$ the following holds.

- If $x \in L$, $\mathbb{P}[A(x) = 1] \geq \frac{2}{3}$.
- If $x \notin L$, $\mathbb{P}[A(x) = 0] \geq \frac{2}{3}$.

Since the measurement of a quantum state is probabilistic in nature, most quantum algorithms come with a certain error probability. It is possible to mitigate an error rate of $\frac{1}{3}$ by independently repeating the algorithm k times and deciding whether $x \in L$ by the majority outcome. As k grows, the error probability decreases.

Since optimization problems are search problems and not decision problems, we define the class of search problems solvable in polynomial time by a quantum Turing machine.

Definition 2.12. The complexity class **FBQP** consists of relations R such that there exists a polynomial-time quantum Turing machine A such that on input $x \in \{0,1\}^*$, A outputs a string $y \in \{0,1\}^*$:

$$\mathbb{P}[(x, y) \in R] \geq \frac{2}{3}$$

An important number-theoretic problem is that of integer factoring where we are tasked with computing a prime factor of a positive integer. This search problem is called search-factoring.

Proposition 2.5 ([20])**.** *search-factoring* $\in$ ***FBQP***

The problem of deciding whether a positive integer has a non-trivial prime factor, is in **P** due to the AKS primality test by Agrawal et al. [2]. We define an alternative version of the factoring problem that is widely believed to not be in **P**. We define

$$\mathsf{factoring} := \{(n,k) \big| n \in \mathbb{N} \text{ has a prime divisor } d \leq k\}$$

There is no known algorithm solving factoring in polynomial time and it is widely believed that there exists no such algorithm. However, this has not been proven. The factoring problem is in **NP**.

Lemma 2.6. *factoring* $\in$ ***NP***

Proof. Let (n,k) be a factoring instance. Let d be a positive integer that will be our candidate witness. We can verify in polynomial time whether d is a prime divisor of n and $d \leq k$. □

An algorithm solving search-factoring in polynomial time can be used to decide factoring in polynomial time by computing the prime factors and checking whether the smallest factor is at most k.

Proposition 2.7. *factoring* $\in BQP$

Proof. Let $N = (n,k)$ be a factoring instance. We decide N using the following algorithm. Use Shor's algorithm [20] to find a candidate factor d of n and check by dividing n with d. If d is a divisor of n, then repeat this step on n/d and n. Else, repeat this step on n. Within a logarithmic number of steps, we have found the prime factorization of n. Finally, we check if the smallest prime factor is at most k. □

Assuming factoring $\notin \mathbf{P}$, this result shows that on factoring, quantum algorithms achieve a super-polynomial speed-up compared its classical variants. In this thesis we show that a similar increase in speed holds for approximation algorithms of MAX-3SAT. We define a quantum approximation algorithm.

Definition 2.13. Let $0 < \alpha \leq 1$, let R be a relation and let m be a target function. A *quantum α-approximation algorithm* for the maximization problem (R,m) is a quantum Turing machine that given an instance $x \in \{0,1\}^*$, outputs a valid solution $y \in S_x$ such that

$$\mathbb{P}[m(x,y) \geq \alpha \operatorname{OPT}(x)] \geq \frac{2}{3}$$

Quantum approximation algorithms for minimization problems are again defined analogously for $\alpha \geq 1$.

In this thesis we show that there exists an optimization problem that admits a quantum approximation algorithm that achieves a better approximation ratio than any classical algorithm.

3 Main theorem

The main result of this thesis is based on a theorem of Szegedy [22].

We prove a slightly different statement than that of Szegedy, who shows that there exists a subset of MAX-3SAT instances such there is no classical $(\frac{7}{8} + 0.01)$-approximation algorithm for this subset, while there is a quantum 0.99-approximation algorithm. We prove that these approximation properties hold for a different maximization problem, namely a maximization problem whose instances are pairs of MAX-3SAT instances and factoring instances. Additionally, we show that this statement is true for any sufficiently small ϵ. In this section we state the theorem and show the outline of the proof that is similar to the proof of the theorem from Szegedy.

Theorem 3.1. *Assuming that* factoring $\notin \boldsymbol{P}$*, for any sufficiently small $\epsilon > 0$ there exists a maximization problem, defined by the pair (R,m) of a relation $R := R_\epsilon$ and a target function m such that*

1. *(Classical inapproximability) There exists no classical polynomial-time $(\frac{7}{8}+\epsilon)$-approximation algorithm for (R,m).*

2. *(Quantum approximability) There exists a polynomial-time quantum* $(1-\epsilon)$*-approximation algorithm for* (R, m)*.*

To prove this theorem, we reduce factoring to GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT via a Karp reduction α' and a gap-producing reduction β', that is, the reduction $\beta \circ \alpha$ reduces factoring to GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT. We show that GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT is **NP**-hard. We will show, as is well known in complexity theory, that along with the assumption that $\mathbf{P} \neq \mathbf{NP}$, this results in the classical inapproximability of MAX-3SAT.

For the quantum approximability, we need to recover the original instance N from an instance $\phi = \beta'(\alpha'(N))$. Using Shor's algorithm, we can construct a witness for N. We show in this thesis that this witness can then be transformed into an assignment that satisfies a $(1-\epsilon) \cdot OPT(\phi)$ clauses of ϕ. To recover the original instance N, we need a polynomial-time algorithm that given a GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT instance ϕ in the image of $\beta' \circ \alpha'$, computes a factoring instance N such that $\phi = \beta'(\alpha'(N))$.

We have not found a polynomial-time algorithm to construct such a pre-image N. We use a workaround that was suggested by Szegedy in an online discussion [1]. The solution is to append the original factoring instance N to the formula $\phi_N = \beta'(\alpha'(N))$, resulting in the pair (ϕ_N, N).

The instances of the relation R will be pairs of GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT instances and factoring instances in the image of $\beta \circ \alpha$. To compute a pre-image of some (ϕ_N, N), we can simply read out N. By letting the target function output the value of the MAX-3SAT target function value of ϕ_N, we preserve the (in)approximability factors from classical and quantum Turing machines.

We define the language 3SAT-N that contains the pairs (ϕ, N) of 3SAT instances and factoring instances such that ϕ is satisfiable. The set of instances for MAX-3SAT-N and GAP[c,s]-3SAT-N contains all pairs (ϕ, N) of 3SAT instances and factoring instances. As described earlier, the target function on input (ϕ, N) outputs the value of the MAX-3SAT target function value of ϕ.

We return to the reduction $\beta' \circ \alpha'$ from factoring to GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT. The first reduction, α', transforms factoring instances into 3SAT instances. This reduction is a direct result of the Cook-Levin theorem [8] [14].

The second reduction, β', is a gap producing reduction from 3SAT to GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT. This reduction actually consists of multiple reductions that can be categorized in two steps.

The first step is introducing a small, but constant gap to the 3SAT instances, meaning that the set has a gap of $(1, c)$, for some constant $c < 1$ that is close to 1. This reduction is the result of the PCP theorem proven by Arora et al. [5]. In this thesis, we go over a simplified proof by Dinur [9]. The second step, proven by Håstad [12], reduces this set to a set of 3SAT instances with a gap $(1-\epsilon, \frac{7}{8}+\epsilon)$, for any sufficiently small ϵ.

We show the reductions in the following graph.

$$\text{factoring} \xrightarrow{\alpha'} \text{3SAT} \xrightarrow{\beta'} \text{GAP}[1-\epsilon, \tfrac{7}{8}+\epsilon]\text{-3SAT}$$

In this thesis, we prove two claims that form the basis of the main theorem. Later in this section we will follow the arguments of Szegedy to show that we can use these claims to prove the main theorem.

Claim 3.2. There exists a Karp reduction α from factoring to 3SAT-N with the following property.

1. Let $N := (n, k) \in$ factoring be a factoring instance and let d be a divisor of n with $d \leq k$. By the definition of a Karp reduction it holds that $(\phi, N) := \alpha(N) \in$ 3SAT-N. On input N and d, we can in polynomial time in $|N| + |d|$ construct a satisfying assignment for ϕ.

Claim 3.3. For any sufficiently small $\epsilon > 0$, there exists a gap-producing reduction $\beta := \beta_\epsilon$ from 3SAT-N to GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT-N with the following properties.

1. Let $(\phi, N) \in$ 3SAT-N and let a be a satisfying assignment for ϕ. By the definition of a gap-producing reduction it holds that $(\psi, N') := \beta(\phi, N) \in L_{yes}$. On input ϕ and a, we can in polynomial time in $|\phi| + |a|$ construct an assignment for ψ that satisfies a $1 - \delta$ fraction of its clauses.

2. Let $(\phi, N) = \beta(\alpha(N))$ be a GAP$[1-\epsilon, \frac{7}{8}+\epsilon]$-3SAT-N instance in the image of $\beta \circ \alpha$. Given (ϕ, N), we can in polynomial time in the size of (ϕ, N) compute a factoring instance (N') such that $(\phi, N) = \beta(\alpha(N'))$.

These reductions will be chained together, resulting in the following chain of reductions.

$$\text{factoring} \xrightarrow{\alpha} \text{3SAT-N} \xrightarrow{\beta} \text{GAP}[1-\epsilon, \tfrac{7}{8}+\epsilon]\text{-3SAT-N}$$

$$N \longmapsto (\alpha'(\phi), N) \longmapsto (\beta'(\alpha'(\phi)), N)$$

Most of the work in thesis consists of laying out the known theory that is used to prove claims 3.2 and 3.3. In section 4.1 we construct a reduction from factoring to 3SAT. In the sections 4.2 and 4.3 we construct a reduction from 3SAT to GAP$[1-\epsilon, \frac{7}{8}+\epsilon]$-3SAT. In section 5, we complete the proof of both claims by adding information on the original factoring instance.

Proof of theorem 3.1. We start the proof assuming the validity of claim 3.2 and claim 3.3. Let α be the Karp reduction from claim 3.2 and let β be the gap-producing reduction from claim 3.3.

$$\Phi^{3CNF}_{factoring} := \{\beta(\alpha(N)) | N \in \{0,1\}^*\}$$

We now define the following relation for our optimization problem.

$$R := \{((\phi_N, N), a) \big| (\phi_N, N) \in \Phi^{3CNF}_{factoring}, a \in \{0,1\}^n : \phi_N \text{ has } n \text{ variables}\}$$

Let m be the function that given $((\phi_N, N), a) \in R$, outputs the number of clauses of ϕ_N satisfied by a.

We show that for the pair (R, m), the two conditions of the theorem hold.

1. (Classical inapproximability) Assume for the sake of contradiction that there exists a classical polynomial-time $(\frac{7}{8}+\epsilon)$-approximation algorithm for (R, m), say A. Given $(\phi, N) := \beta \circ \alpha(N)$, for some factoring instance N, this algorithm outputs an assignment that satisfies at least a $(\frac{7}{8}+\epsilon) \cdot \text{OPT}(\phi)$ fraction of clauses of ϕ. We show how this algorithm can be used to decide the problem factoring in polynomial time. Let $N = (n, k)$ be an arbitrary factoring instance and define the 3SAT-N instance $(\phi, N) := \beta(\alpha(N))$. The algorithm A can find an assignment a of ϕ such that at least a $\frac{7}{8}+\epsilon$ fraction of the optimal number of clauses are satisfied.

   - If $(n, k) \in$ factoring, ϕ is at least $(1-\delta)$-satisfiable. This means that a satisfies at least a $(1-\delta)(\frac{7}{8}+\epsilon)$ fraction of clauses of ϕ.
   - If $(n, k) \notin$ factoring, ϕ is at most $(\frac{7}{8}+\delta)$-satisfiable. In this case a satisfies at most $\frac{7}{8}+\delta$ of all clauses of ϕ.

   Note that the value of δ is not fixed. We choose δ such that $(1-\delta)(\frac{7}{8}+\epsilon) > \frac{7}{8}+\delta$, any $0 < \delta < \frac{8\epsilon}{8\epsilon+15}$ suffices. This choice lets us distinguish between "yes-instances" and "no-instances". If (n, k) is a "yes-instance", a satisfies strictly more than a $\frac{7}{8}+\delta$ fraction and if it is a "no-instance", a satisfies at most a $\frac{7}{8}+\delta$ fraction of the clauses of ϕ.

   By evaluating the assignment a on ϕ and checking the number of clauses it satisfies, we can in polynomial time decide whether N is a "yes-instance" or a "no-instance", contradicting the assumption that factoring $\notin \mathbf{P}$. We conclude that for any $\epsilon > 0$ there exists no polynomial-time $(\frac{7}{8}+\epsilon)$-approximation algorithm for (R, m), unless factoring $\in \mathbf{P}$.

2. (Quantum approximability) Let $(\phi_N, N) \in \Phi^{3CNF}_{factoring}$. By property 2 of claim 3.3, we can find N' such that $(\phi_N, N) = \beta(\alpha(N'))$. Note that this N' is precisely $N = (n, k)$. Using Shor's algorithm, we decide in polynomial time whether $N \in$ factoring and we distinguish between two cases.

- If $N \in$ factoring, by proposition 2.5, we can in polynomial time compute a positive integer $d \leq k$ that is a divisor of n. By property 1 of both claims, we can transform this witness d into a witness a of MAX-3SAT that satisfies a $1-\delta$ fraction of the clauses of ϕ. Since $\delta < \epsilon$ and $\mathrm{OPT}(\phi)$ is at most the number of clauses in ϕ, this assignment satisfies a $1-\epsilon$ fraction of $\mathrm{OPT}(\phi)$. We conclude that our algorithm is a quantum $(1-\epsilon)$-approximation algorithm for (R, m).
- If $N \notin$ factoring, ϕ is at most $(\frac{7}{8}+\delta)$-satisfiable. The algorithm from lemma 2.3 outputs an assignment that satisfies at least a $\frac{7}{8}$ fraction of clauses of ϕ. Since $\frac{7}{8}/(\frac{7}{8}+\delta) \geq 1-\epsilon$ holds when $\delta < \frac{8\epsilon}{8\epsilon+15}$, this assignment satisfies at least a $1-\epsilon$ fraction of $\mathrm{OPT}(\phi) \leq \frac{7}{8}+\delta$. So, this algorithm is a quantum $(1-\epsilon)$-approximation algorithm for (R, m). □

We introduce a formalization of the first part of both claims. Namely, a *witness transformation*. The idea of transforming one witness into another is not new. Many constructions of Karp reductions explicitly state how the solution of one problem can be transformed into the solution of another. In this thesis we introduce a formal definition of this idea that will be used to chain multiple witness transformations together.

First, we remind the reader that every language L in **NP** has a polynomial-time verifier and a corresponding relation R_L. We consider promise problems whose "yes-instances" are also polynomial-time verifiable. That is, promise problems $L = (L_{yes}, L_{no})$ for which there exist a polynomial-time verifier V such that for any $x \in \{0,1\}^*$ the following holds.

- If $x \in L_{yes}$, then there exists a witness w such that $V(x, w) = 1$.
- If $x \in L_{no}$, then $V(x, w) = 0$ for every witness w.

For a polynomial-time verifiable promise problem L, we define the relation

$$R_L := \{(x, w) \big| V(x, w) = 1\}$$

Let L, L' be polynomial-time verifiable languages or promise problems such that there exists a Karp reduction f from L to L'. That is, L can be a language or a promise problem and so can L'.

Definition 3.1. *A witness transformation* of f is a polynomial-time computable function $g : R_L \to \{0,1\}^*$ such that for any pair of a "yes-instance" and a witness $(x, w) \in R_L$, we have that $(f(x), g(x, w)) \in R_{L'}$.

Property 1 of claim 3.2 and 3.3 states that such witness transformations exist for the given reductions. Karp reductions are closed under composition, but witness transformations are not, because two witness transformations do not directly compose. However, witness transformations can still be chained together by carrying over information on the instance.

Lemma 3.4. *Let L_1, L_2, L_3 be polynomial-time verifiable languages or promise problems such that there exist Karp reductions f_1 from L_1 to L_2 and f_2 from L_2 to L_3 that both admit a witness transformation. Then the Karp reduction $f_2 \circ f_1$ admits a witness transformation.*

Proof. Let $g_1 : R_{L_1} \to \{0,1\}^*$ and $g_2 : R_{L_2} \to \{0,1\}^*$ be witness transformations for f_1 and f_2 respectively. Let $(x, w) \in R_{L_1}$, since g_1 is a witness transformation it holds that

$$(f_1(x), g_1(x, w)) \in R_{L_2}$$

and by the witness transformation property of g_2, it holds that

$$(f_2(f_1(x)), g_2(f_1(x), g_1(x, w))) \in R_{L_3}$$

In other words, the function $g : R_{L_1} \to \{0,1\}^*$ given by $(x, w) \mapsto g_2(f_1(x), g_1(x, w))$ is a witness transformation for $f_2 \circ f_1(x)$. □

In the following sections, we prove that the polynomial-time reductions and polynomial-time witness transformations from claim 3.2 and 3.3 exist. We do this in multiple steps, constructing a sequence of reductions and witness transformations and chaining them together in section 5.

4 Reductions

In this section we present the chain of reductions from factoring to GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT.

In subsection 4.1 we reduce factoring instances into 3SAT instances. This reduction is a direct result of the Cook-Levin theorem [8] [14].

In subsection 4.2 we introduce a gap to the 3SAT instances: we reduce 3SAT to GAP[1,c]-3SAT, for some constant $0 < c < 1$. The existence of this reduction is a direct consequence of the PCP theorem [5]. We use the reduction that is given in the simplified proof of the PCP theorem by Dinur [9].

The gap introduced in subsection 4.2 is increased to the optimal size in subsection 4.3: we reduce the problem GAP[1,c]-3SAT to GAP[$1-\epsilon, \frac{7}{8}+\epsilon$]-3SAT. This reduction is given by Håstad [12] in his proof that MAX-3SAT is not approximable with a factor greater than $\frac{7}{8}$.

Additionally, we prove that there exist witness transformations for these reductions.

4.1 Cook-Levin theorem

The first reduction of our chain follows from the Cook-Levin theorem. The theorem states that SAT is **NP**-complete, meaning that SAT $\in$ **NP** and any language in **NP** is Karp-reducible to SAT. We then reduce SAT to 3SAT. We follow the proof of the Cook-Levin theorem as presented in *Introduction to the Theory of Computation* by Sipser [21, Chapter 7], but we make a slight change that will be made precise later.

To show that for each **NP** language $L \in$ **NP**, there exists a Karp reduction from L to SAT, Cook and Levin analyze a non-deterministic Turing machine that decides L. We do not define what a non-deterministic Turing machine, because we will work with a deterministic version that allows for an easier construction of the witness transformation.

We change the proof of the Cook-Levin theorem by considering a deterministic verifier. Let L be a language and let $V = V_L$ be a deterministic verifier for L. Let $x \in \{0,1\}^*$ be an instance and let $w \in \{0,1\}^*$ be a candidate witness. We analyze V on the input (x, w) and express the inner mechanics in terms of a boolean formula.

First, we introduce a string that represents the state of a Turing machine at some step of computation. Consider a Turing machine M in some computation step where the symbols in the tape are given by $x_1, x_2, \ldots, x_n, \sqcup, \sqcup, \ldots$ and each symbol after x_n is the empty symbol. The read/write head is at x_i and the current state symbol is q_j. We represent the state of M with the following string:

$$x_1 x_2 \ldots x_{i-1} q_j x_i \ldots x_n$$

We call this string the *configuration* of M.

We are now in the position to prove the Cook-Levin theorem.

Theorem 4.1. *SAT is **NP**-complete.*

Proof. By lemma 2.1, SAT is in **NP**.

Next, we prove that for any **NP** language L, there exists a Karp reduction from L to 3SAT. Let V_L be a verifier of L. Our reduction takes an instance $x \in \{0,1\}^*$ as input and describes the inner workings of V_L on x and a witness. This description will be written in terms of a 3CNF formula such that the formula is satisfiable if and only if there exists a witness causing V_L to accept.

Let $L \in$ **NP** and let $x \in \{0,1\}^*$ with size $n := |x|$. Let $V := V_L$ be a verifier for L with state set Q, tape alphabet Σ, input alphabet Γ and transition function δ. Since V runs in polynomial time, given input

x and for some $k \in \mathbb{N}$, V makes at most $N := n^k$ computational steps for sufficiently large n. We are usually interested in the asymptotic behavior of the time complexity of a reduction. For this reason, it suffices to assume that V runs time time n^k.

We now define a *tableau* for V on input (x, w), for some $w \in \{0,1\}^*$ that will represent the calculations done by V on input (x, w). A tableau for V on input (x, w) is a $n^k \cdot n^k$ table of symbols in $C := Q \cup \Sigma \cup \{\#\}$. Each row of a tableau corresponds to a configuration of V. We write tableau$[i, j] = s$ if the symbol in position (i, j) of the tableau is equal to $s \in C$.

To enforce that a tableau correctly represents the computation of V, we will require it to have the following properties:

- The first row is the following concatenation: $\#|x|w|\#$. Without loss of generality, we assume that w is of size $|w| = n^k - n - 2$. This is equal to the maximum number of bits read from w, since V runs in time n^k. If w is smaller, we can add $n^k - n - 2 - |w|$ characters $\sqcup'$ to the end that the Turing machine will interpret as the empty symbol $\sqcup$.
- For convenience later in the proof, we define the left- and rightmost characters of every row to be $\#$.
- Besides the first row, a row can only represent the configuration that follows from the transition function and the configuration represented by the row above.

We call a tableau satisfying these properties an *accepting* tableau if any of the rows contain the accepting state q_{accept}. An accepting tableau represents an accepting internal computation of V. In other words, V accepts on input (x, w) if and only if there exists an accepting tableau for V on (x, w).

We will construct a 3CNF formula $\phi = \phi_{V,x}$ that is satisfiable if and only if there exists some $w \in \{0,1\}^*$ such that there exists an accepting tableau for V on (x, w). We introduce the $|C|(n^k)^2$ variables $x_{i,j,s}$, with $s \in C$ and $1 \leq i, j \leq n^k$. The formula ϕ will correspond to a tableau. The variable $x_{i,j,s}$ being equal to 1 corresponds to position (i, j) of the tableau containing s.

We construct ϕ such that it checks four different aspects of the corresponding tableau:

- Each position in the tableau contains exactly 1 symbol,
- the first row is the state of V on input x and w,
- the rest of the configurations follow from the first row by the rules of the transition function
- and the tableau is an accepting tableau.

We construct four boolean formulas, ϕ_{cell}, ϕ_{start}, $\phi_{move,3CNF}$ and ϕ_{accept} corresponding to these four items. A formula is only satisfied when the tableau satisfies the corresponding item.

These formulas are not in 3CNF, but they are structured such that it is easy to transform them into 3CNF.

We first enforce that each position in the tableau contains exactly one symbol. We define

$$\phi_{cell} := \bigwedge_{1 \leq i,j \leq n^k} \left[\Big(\bigvee_{s \in C} x_{i,j,s} \Big) \wedge \Big(\bigwedge_{s,t \in C, s \neq t} \overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \Big) \right]$$

This formula equates to true if and only if for all cells, there is at least one symbol is this cell and for each pair of symbols, they are not both contained in the cell.

Next, we construct a formula that enforces that the first row of a tableau is the starting state of V on input (x, w). We achieve this by constructing two formulas.

First, the head starts at the first symbol and V is in state q_0. The first n symbols after q_0 are the symbols of x. We also enforce that the first and last symbols are #.

$$\begin{aligned}\phi_{start,base} := & \; x_{1,1,\#} \wedge \\ & \; x_{1,2,q_0} \wedge \\ & \; x_{1,3,x_1} \wedge \; \ldots \; \wedge x_{1,n+2,x_n} \wedge \\ & \; x_{1,n^k,\#}\end{aligned}$$

Second, for the rest of the variables in the first row (except for the last #), we disallow symbols that are not input symbols or the empty symbol. Once an empty symbol has been written, all symbols to the right of it are the empty symbol. This ensures that we do not use the empty symbol as part of a witness.

$$\phi_{witness} := \bigwedge_{s \in C \setminus (\Gamma \cup \{\sqcup\})} \overline{x_{1,n+3,s}} \wedge \; \ldots \wedge \bigwedge_{s \in C \setminus (\Gamma \cup \{\sqcup\})} \overline{x_{1,n^k-1,s}} \bigwedge_{n+2<j<n^k-1} \overline{x_{1,j,\sqcup}} \vee x_{1,j+1,\sqcup}$$

We now define

$$\phi_{start} := \phi_{start,base} \wedge \phi_{witness}$$

The following requirement is that the tableau is an accepting one. For this, we simply require the symbol q_{accept} to appear at least once in the tableau.

$$\phi_{accept} := \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{accept}}$$

Finally, we require each row to represent a configuration that follows from the transition function and the previous row. First, we define an (i,j)-*window* to be the set of cells

$$\begin{aligned}\big\{&\text{tableau}[i,j-1],\ \text{tableau}[i,j],\ \ \text{tableau}[i,j+1],\\ &\text{tableau}[i+1,j-1],\ \text{tableau}[i+1,j],\ \ \text{tableau}[i+1,j+1]\big\}\end{aligned}$$

We say that an (i,j)-window is a *legal* window if the values of tableau$[i+1,j-1]$, tableau$[i+1,j]$ and tableau$[i+1,j+1]$ follow from the values of tableau$[i,j-1]$, tableau$[i,j]$ and tableau$[i,j+1]$ according to the transition function. We now construct a boolean formula that enforces that an $(i,j)-$window is a legal window. We construct the formula as follows.

$$\phi_{(i,j)\text{-window is legal}} := \bigvee_{a_1,\ldots,a_6 \text{ is a legal window}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

We require every possible (i,j)-window to be legal. It is not difficult to convince ourselves that if for some fixed row number $1 \le i^* < n^k$ all (i^*,j)-windows are legal, then row i^*+1 follows from the transition function and row i^*. We leave the details to the interested reader. We construct a formula that reflects this requirement for each row.

$$\phi_{move} := \bigwedge_{1 \le i < n^k, 1 < j < n^k} \phi_{(i,j)\text{-window is legal}}$$

We have created four formulas that each enforce a different aspect of an accepting tableau. Define the formula ϕ as follows.

$$\phi := \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}$$

By construction, this formula is satisfiable if and only if there exists some w such that $V(x,w) = 1$. In other words, if and only if $x \in L$.

We now give some motivation that this process runs in polynomial time. First, we argue that the size of ϕ polynomial in n.

The formula ϕ consists of $|C|n^{2k}$ variables. So, writing a variable takes up $O(log(n))$ space. We consider the size of each sub-formula.

First, ϕ_{cell} has a block of constant size for each of the $O(n^{2k})$ variables. The formula ϕ_{start} is of size $O(n^k)$. Next, ϕ_{accept} contains exactly one variable for each of the $O(n^{2k})$ locations of the tableau. Finally, note that for each (i, j), there are at most $|C|^6$ different legal (i,j)-windows, so the formula $\phi_{(i,j)\text{-window is legal}}$ is of constant size. Since there are $O(n^{2k})$ windows, ϕ_{move} is of size $O(n^{2k})$.

We do not go into detail as to why this reduction runs in polynomial time. The fact that ϕ is of polynomial size should gives us a good intuition as to why it can be constructed in polynomial time. □

We now show that every boolean formula constructed in this manner can be reduced to a 3CNF formula, thus showing that 3SAT is also **NP**-hard.

Theorem 4.2. *3SAT is **NP**-complete.*

Proof. By lemma 2.1, SAT is in **NP**.

Let $L \in \mathbf{NP}$ be a language, we show that we can construct a Karp reduction from L to 3SAT. Let $x \in \{0,1\}^*$ be an instance and let ϕ_{cell}, ϕ_{start}, ϕ_{move} and ϕ_{accept} be the formulas as constructed in the proof of theorem 4.1. Due to their structure, each formula can be transformed into 3CNF formulas in polynomial time.

For ϕ_{cell}, we observe that by the distributive rules of disjunctions and conjunctions, we can arrange the variables and gates such that ϕ_{cell} is in kCNF for some $k \in \mathbb{N}$.

We transform this kCNF formula into a 3CNF formula. Let $C := l_1 \vee l_2 \vee \ldots \vee l_k$ be a clause of ϕ_{cell}, where l_i is a literal, so either x_i or its negation. We replace C with the clauses $l_1 \vee l_2 \vee y_1$, $\overline{y_1} \vee l_3 \vee y_2$, $\ldots$, $\overline{y_{n-3}} \vee l_{n-1} \vee l_n$, where y_i is a new variable. This set of clauses is satisfiable if and only if C is satisfiable. We perform this process for each clause of ϕ_{cell} to construct the 3CNF formula $\phi_{cell,3CNF}$ that is satisfiable if and only if ϕ_{cell} is satisfiable.

This process can be used to transform the other formulas into 3CNF formulas. Let $\phi_{start,3CNF}$, $\phi_{move,3CNF}$ and $\phi_{accept,3CNF}$ be the resulting 3CNF formulas. Our final formula is the conjunction of these four formulas.

$$\phi := \phi_{cell,3CNF} \wedge \phi_{start,3CNF} \wedge \phi_{move,3CNF} \wedge \phi_{accept,3CNF}$$

This formula is satisfiable if and only if there exists some w such that $V(x, w) = 1$. In other words, if and only if $x \in L$.

Each clause is replaced with a clause that is larger by a constant factor. This process can be performed in polynomial time in the size of x. □

We show that the reduction described in theorem 4.2 has an accompanying witness transformation.

Lemma 4.3. *Let $L \in$ **NP** be a language and let α_L be the reduction from theorem 4.2. There exists a polynomial-time witness transformation for α_L.*

Proof. Let $V = V_L$ be the verifier for L that was used to construct α_L. Let $x \in L$, and let w be a witness for x. Run V on input (x, w) and fill in the accepting tableau for V on (x, w). The symbols in the accepting tableau give us an assignment to the variables $x_{i,j,s}$ that satisfies $\alpha_L(x)$. This assignment is a witness for $\alpha_L(x)$. This transformation runs in polynomial time in $|x|$ and thus in polynomial time in $|(x, w)|$. □

In this thesis we are specifically interested in the reduction $\alpha := \alpha_{\mathsf{factoring}}$. Let $(n, k) \in$ factoring and let $d \leq k$ be a divisor of n. By lemma 4.3 we can transform d into a satisfying assignment of $\alpha(n, k)$ in polynomial time. By lemma 4.3, we have an accompanying polynomial-time witness transformation for α.

4.2 PCP theorem

We have shown that 3SAT is **NP**-complete, meaning that we can reduce any language in **NP** to 3SAT. The PCP theorem, first proven by Arora et al. [5] and later simplified by Dinur [9], reduces this set of 3SAT instances to a set of 3SAT instances with a gap of $(1, \alpha)$, for some $0 < \alpha < 1$. Equivalently, the PCP theorem states that for every **NP** language, there exists a *probabilistically checkable proof* verifier.

4.2.1 Probabilistically checkable proofs

The PCP theorem states that we can construct a specific verifier for each language in **NP**, called a verifier in a probabilistically checkable proof system (PCP verifier).

This verifier is a *probabilistic* Turing machine. Besides to the tools of a deterministic Turing machine, a probabilistic Turing machine has (read-only) access to a string $r \in \{0,1\}^*$. Because we are only interested in polynomial-time machines, given an input $x \in \{0,1\}^*$, we limit the size of r to $p(|x|)$, for some polynomial p.

At the start of computation of a probabilistic Turing machine, a string r of size $p(|x|)$ is given uniformly at random. This allows the machine to make random decisions by reading the bits of r.

The PCP verifier also has oracle access to a function $\pi : \{0,1\}^* \to \{0,1\}$, which we will call an *oracle*. A Turing machine M that has oracle access to π can query the value $\pi(x)$ in a single computational step, for any $x \in \{0,1\}^*$. We write M^π for a (deterministic or probabilistic) Turing machine M with query access to π.

We write $\mathbb{P}[M^\pi(x) = b]$ for the probability, taken over the random strings r, that M^π outputs b.

Definition 4.1. Let $q, r : \mathbb{N} \to \mathbb{N}$ be functions. The complexity class $PCP[q(n), r(n)]$ contains the languages L for which there exists a probabilistic polynomial-time Turing machine V such that for any input $x \in \{0,1\}^*$, V makes at most $q(|x|)$ oracle queries, uses $r(|x|)$ random bits and satisfies the following properties.

- (Completeness) If $x \in L$, then there exists an oracle π such that $\mathbb{P}_r[V^\pi(x) = 1] \geq 1$.
- (Soundness) If $x \notin L$, then for any oracle π we have that $\mathbb{P}_r[V^\pi(x) = 1] \leq \frac{1}{2}$.

The PCP theorem states that every language in **NP** is recognized by a PCP verifier with constant query complexity and logarithmic randomness.

Theorem 4.4 (PCP theorem [5])**.** $\boldsymbol{NP} = PCP[O(1), O(log(n)]$

We prove an equivalent version of the PCP theorem that is useful for our purposes.

Theorem 4.5 (PCP theorem)**.** *For some $0 < \alpha < 1$, there exists a reduction from 3SAT to GAP[1,α]-3SAT.*

We refer the reader to [9] for a proof of this equivalence.

In this section we follow the proof of the PCP theorem as given by Dinur [9]. At the end of this section we provide a short proof stating one way of the equivalence, showing that theorem 4.4 follows from theorem 4.5.

4.2.2 Constraint graphs

Dinur uses the problem of satisfying *constraint graphs* to prove the PCP theorem. The idea is to reduce each language in **NP** to a set of constraint graphs and introduce a constant sized gap to this set. This set with a gap can then be reduced to 3SAT with a gap.

Definition 4.2. A *constraint graph* is a tuple $G = ((V, E), \Sigma, \mathcal{C})$ such that

- (V, E) is an undirected graph,
- Σ is a finite alphabet,

- For each edge $e \in E$, there is a *constraint* $c(e) \subseteq \Sigma \times \Sigma$ and $\mathcal{C} = \{c(e)\}_{e \in E}$.

We call (V, E) the *underlying graph* of the constraint graph G and often write G for either. We say that a constraint $c(e)$ is satisfied by $(a, b) \in \Sigma \times \Sigma$ if $(a, b) \in c(e)$. An assignment for a constraint graph G is a mapping $a : V \to \Sigma$. We say that this assignment satisfies G if for each $e = (u, v) \in E$, $(a(u), a(v) \in c(e)$. In words, an assignment satisfies G if the assignment satisfies each constraint. If such an assignment exists, we say that G is satisfiable.

Definition 4.3. The language CG is the set of satisfiable constraint graphs.

We write $F_a := \{(u, v) \in E : (a(u), a(v)) \notin c(e)\}$ for the set of edges whose constraints are not satisfied by some assignment a.

Definition 4.4. For any constraint graph $G = ((V, E), \Sigma, \mathcal{C})$ and assignment a, we define

$$\text{UNSAT}_a(G) := \frac{|F_a|}{|E|}$$

$$\text{UNSAT}(G) := \min_a \text{UNSAT}_a(G)$$

We call this the the unsat-value or simply the unsat of G. Dinur defines the unsat value to analyze the gap of a set of constraint graphs. We use the unsat-value to define a gap version of CG.

Definition 4.5. Let $0 < \alpha < 1$. GAP$[1, \alpha]$-CG is the pair of constraint graphs with UNSAT $= 0$ and the constraint graphs with UNSAT$(G) \geq 1 - \alpha$.

Dinur shows that we can increase the gap size of a set of constraint graphs to a constant size, while only increasing the size of the constraint graphs by a polynomial factor. Let $G = ((V, E), \Sigma, \mathcal{C})$ be a constraint graph, we define the size of G as follows.

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

The alphabet will remain bounded during the reduction and the size of the constraint set is linear in the size of the edge set. So, the actual size of a constraint graph G will always be a constant factor of $size(G)$.

Dinur shows proves the following theorem from which the PCP theorem follows.

Theorem 4.6. *For some $0 < \alpha' < 1$, there exists a reduction from 3SAT to GAP$[1, \alpha']$-CG.*

To prove this theorem, Dinur first introduces a set of constraint graphs that has a gap size dependent on the size of the constraint graph. The problem of 3-coloring has a gap size that is a function of the graph size. We say that a graph $G = (V, E)$ is *3-colorable* if there exists a function $c : V \to \{1, 2, 3\}$ such that for all $(u, v) \in E$, $c(u) \neq c(v)$.

Definition 4.6. The language 3-coloring is the set of 3-colorable graphs.

Lemma 4.7. *3-coloring is NP-complete.*

Proof. Given a graph $G = (V, E)$ and a function $c : V \to \{1, 2, 3\}$, we can in polynomial time check whether c satisfies $c(u) \neq c(v)$ for all $(u, v) \in E$. So, 3-coloring is in NP. Due to Karp [13], we can reduce 3SAT to 3-coloring and 3-coloring is **NP**-complete. □

Lemma 4.8. *Let β be the reduction from lemma 4.7. There exists a polynomial-time witness transformation for β.*

Proof. The witness transformation follows directly from the reduction and can clearly be computed in polynomial time. □

Lemma 4.9. *Given a constraint graph $G = ((V, E), \Sigma, \mathcal{C})$ with $|\Sigma| = 3$, it is **NP**-hard to decide whether UNSAT$(G) = 0$ or UNSAT$(G) \geq \frac{1}{|G|}$.*

Proof. Let G' be a 3-coloring graph and let $\Sigma = \{1, 2, 3\}$ be the 3 different colors. We define the constraint graph $G = ((V, E), \Sigma, \mathcal{C})$ as follows. Let the underlying graph of G be the graph G' and set the constraint of an edge (u, v) to be the values where u and v are unequal. This reduction is a Karp reduction.

The set of constraint graphs resulting from this reduction has a gap that is dependent on the size of the graph. The unsat value of a graph with unsat value unequal to 0 is at least $\frac{1}{|E|} > \frac{1}{|G|}$. □

Lemma 4.10. *Let β be the reduction from lemma 4.9. There exists a polynomial-time witness transformation for β.*

Proof. Let $G \in$ 3-coloring be a 3-colorable graph, let $G' = \beta(G)$ be the constraint graph resulting from the reduction and let $c : V \to \{1, 2, 3\}$ be a coloring for G. By the construction of G', this coloring satisfies the constraints of G'. □

Lemma 4.9 shows that we can reduce any **NP** problem to a set of constraint graphs with a gap of $(1, 1 - \frac{1}{|G|})$. This set will be reduced to a set of constraint graphs with a gap of $(1, \alpha')$, for some constant α'. In the following subsection, we present the methods used by Dinur to achieve this constant gap.

4.2.3 Proof overview

The important contribution of Dinur is the amplifying reduction. This step increases the unsat value of a constraint graph by a constant factor t. It does so by creating new edges that represent a walk of t edges. Dinur proves that such an edge checks $O(\sqrt{t})$ different constraints at once. An edge not satisfied by an assignment in the original graph results in $O(\sqrt{t})$ edges not being satisfied in the amplified graph.

To apply the amplifying step, the graph requires a fixed structure. In the first section we transform the graph into a graph fitting for the amplifying step, while only inducing a linear blowup in the size and a constant reduction of the unsat value.

The amplifying step does however increase the alphabet size. To solve this issue, we replace a constraint with a smaller constraint graph that simulates this constraint over a smaller alphabet.

Combining all steps results in doubling the unsat value of a constraint graph. Repeating this process a logarithmic number of times finally results in a constant unsat value.

4.2.4 Preprocessing

The effectiveness of the amplifying step relies on the structure of the constraint graph. The underlying graph should have constant degree and a self-loop on each vertex. Besides these properties, the graph should have a large enough expansion rate, meaning that two independent subset of nodes are connected with a high number of edges.

Given a graph $G = \{V, E\}$ and two subsets of vertices $S, T \subset V$ we denote $E(S, T) := |S \times T \cap E|$ for be the number of edges between S and T. For a subset of nodes $S \subset V$ we denote $\overline{S} := V \setminus S$ for the complement of S.

Definition 4.7. The edge expansion rate of a d-regular graph $G = \{V, E\}$ is defined as:

$$h(G) := \min_{S:|S| \leq |V|/2} \frac{E(S, \overline{S})}{|S|}$$

We use a result that shows there exists a *family of expander graphs.*

Lemma 4.11 ([9])**.** *There exists constants $d_0 \in \mathbb{N}$ and $h_0 > 0$ such that there is a polynomial-time constructable set (family) of $\{X_n\}_{n \in \mathbb{N}}$ of d_0-regular graphs $X_n = (V_n, E_n)$, with $|V_n| = n$ and $h(X_n) \geq h_0$.*

We give some definitions and prove some technical lemmas to aid with making a constraint graph have the right expansion properties.

Definition 4.8. Let $G = (V, E)$ be a (multi)graph with $V = \{v_1, v_2, \ldots, v_n\}$. We define the $n \cdot n$ adjacency matrix A_G, or just A, by setting $A_{ij} = k$ if there are k edges connecting v_i and v_j.

Definition 4.9. Let $A = A_G$ be the adjacency matrix of some graph G and let $\lambda_1, \lambda_2, \ldots, \lambda_{n-1}$ be the spectrum of A. We define $\lambda(G) := \max(\lambda_2, |\lambda_{n-1}|)$.

This is the second largest eigenvalue in absolute value. This value is closely related to the expansion rate of the graph, as shown by the following technical lemma's.

Lemma 4.12 ([9])**.** *Let $A = A_G$ be the adjacency matrix of some graph G, then*

$$\lambda(G) = \max_{x \neq \vec{0}, \langle x, \vec{1} \rangle = 0} \frac{|\langle x, A_{G'} x \rangle|}{|\langle x, x \rangle|}$$

Lemma 4.13 ([9])**.** *Let G be a d-regular graph and let A be its adjacency matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. Then*

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

We use these lemmas to construct a family of regular expander graphs with λ value.

Lemma 4.14. *There exists constants $d_0' \in \mathbb{N}$ and $0 < \lambda_0 < d_0'$, such that there is a polynomial-time constructable family $\{G_n\}_{n \in \mathbb{N}}$ of d_0'-regular graphs $G_n = (V_n, E_n)$, with $|V_n| = n$ and $\lambda(G_n) \leq \lambda_0$.*

Proof. We use lemmas 4.11 and 4.13 to construct a graph with the desired properties. For any $n \in \mathbb{N}$, let $X_n = (V_n, E_n)$ be the d_0-regular graph from lemma 4.11. Let E_n^1, be the set of d_0 self loops on each vertex $v \in V_n$. Define $E_n' = E_n \cup E_n^1$ and $G_n = (V_n, E_n')$, a $d_0' = 2d_0$-regular graph with $|V_n| = n$. Since there are no vertices and no edges between vertices added, $h(G_n) = h(X_n) \geq h_0$.

Note that if $\lambda_i < -d_0$ is an eigenvalue with eigenvector v_i of X_n, then $\lambda_i' = \lambda_i + d_0 > 0$ is an eigenvalue of the same eigenvector of G_n. This means that all eigenvalues of G_n are positive, so now we have that $\lambda(G_n) = \lambda_2$. Let $\lambda_0 = d_0' - \frac{(h_0)^2}{2d_0'} < d_0'$, now $\lambda(G_n) = \lambda_2 \leq d_0' - \frac{h(G_n)^2}{2d_0'} \leq d_0' - \frac{(h_0)^2}{2d_0'} = \lambda_0$. □

We are now in the position to introduce the first preprocessing step. This step consists of transforming a graph into a graph with constant degree. We achieve this by, for each original vertex v, adding a "cloud" of vertices with one vertex for each edge incident to v. We connect the vertices of a cloud with an expander graph with no constraints. We also connect each pair of clouds with one edge if their original vertices were connected, these edges inherit the constraint that was on the original edge.

Definition 4.10 (Preprocessing step 1)**.** Let $G = ((V, E), \Sigma, \mathcal{C})$ be a constraint graph, we define $G' := prep_1(G) := ((V', E'), \Sigma, \mathcal{C}')$ as follows:

- For each $v \in V$, we define $[v] := \{(e, v) \mid e \in E \text{ is incident on } v\}$ and we define $V' := \bigcup_{v \in V} [v]$.
- For each $v \in V$, let $X_v = (V_v, E_v)$ be a d_0'-regular graph on $[v]$ with edge expansion at least h_0, whose existence is guaranteed by lemma 4.11. Let

  $$E_1 := \bigcup_{v \in V} E_v$$

  and let

  $$E_2 := \{((e, v), (v', e)) | (v, v') \in E\}$$

  Finally, we define $E' := E_1 \cup E_2$.
- For each $e' \in E$ we define the constraints as follows.
  - If $e' \in E_1$, $c(e') = \{(\sigma, \sigma) | \sigma \in \Sigma\}$.
  - If $e' = ((v, e), (v', e)) \in E_2$, $c(e') = c(e) \in \mathcal{C}$.

Dinur shows that this process results in the required properties and only changes the graph size and the unsat value by a constant factor.

Lemma 4.15. *There are constants $d_0, c > 0$ such that for every constraint graph $G = ((V, E), \Sigma, \mathcal{C})$, the constraint graph $G' = prep_1(G) = ((V', E'), \Sigma, \mathcal{C}')$ is $(d_0 + 1)$-regular, with $Size(G) = O(Size(G'))$ and has the properties that $|V'| \leq 2|E|$ and*

$$c \cdot UNSAT(G) \leq UNSAT(G') \leq UNSAT(G)$$

Proof. Let d_0' be the constant from lemma 4.14. G' is $(d_0' + 1)$-regular. For every edge in G, there are two vertices in G', or one in the case of a self-loop. So clearly $|V'| \leq 2|E|$. We now also have that $Size(G') = |E'| + |V'| \leq d_0' \cdot |V| + |E| + 2|E| = O(Size(G))$.

Next, we show that $\text{UNSAT}(G') \leq \text{UNSAT}(G)$ by showing that an assignment $a : V \to \Sigma$ can be used to construct an assignment $a' : V' \to \Sigma$ such that $\text{UNSAT}_{a'}(G') \leq \text{UNSAT}_a(G)$. Let a be an assignment such that $\text{UNSAT}_a(G) = \text{UNSAT}(G)$ and let $a' : V' \to \Sigma$ be the assignment $(v, e) \mapsto a(v)$. This new assignment satisfies $c(e)$ for all $e \in E_1$ and satisfies exactly $|E_2| \cdot (1 - \text{UNSAT}(G))$ of the constraints on E_2. The following holds.

$$\text{UNSAT}(G') \leq \text{UNSAT}_{a'}(G') = \frac{\text{UNSAT}_a(G)|E_2|}{|E_1| + |E_2|} \leq \frac{\text{UNSAT}_a(G)|E_2|}{|E_2|} = \text{UNSAT}_a(G) = \text{UNSAT}(G)$$

We now show that there exists some global constant c such that $c \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G')$ by showing that an assignment $a' : V' \to \Sigma$ can be used to find an assignment $a : V \to \Sigma$ such that $c \cdot \text{UNSAT}_a(G) \leq \text{UNSAT}_{a'}(G')$. Let $a' : V' \to \Sigma$ be an assignment. For every $v \in V$, we define an assignment for a vertex v by taking the most "popular" opinion on v by the assignment a':

$$a(v) := \arg\max_{\sigma \in \Sigma} |\{(v, e) \in [v] : a'(v, e) = \sigma\}|$$

Let $d = d_0' + 1$ and note that $|E| \leq d|E'|$. Fix some assignment $a' : V' \to \Sigma$ and define $a : V \to \Sigma$ according to (4.2.4). Let $F := F_a \subseteq E$ and $F' := F'_{a'} \subseteq E'$ be the set of edges not satisfied by a and a'. Let $S \subseteq V'$ be the set of vertices that are assigned a different value than the most popular value by a':

$$S := \bigcup_{v \in V} \{(v, e) \in [v] : a'(v, e) \neq a(v)\}$$

For $e \in F$ we have that either $((v, e)(v', e)) \in F'$ or at least one of v and v' is in S. This means that $|F'| + |S| \geq |F| = \text{UNSAT}_a(G)|E|$. We now separate two cases:

- If $|F'| \geq \frac{\text{UNSAT}(G)|E|}{2}$, then $\text{UNSAT}_{a'}(G') = \frac{|F'|}{|E'|} \geq \frac{\text{UNSAT}_a(G)|E|}{2|E'|} \geq \frac{\text{UNSAT}_a(G)|E|}{2d|E|} = \frac{1}{2d}\text{UNSAT}_a(G)$ and we are done.
- If $|F'| < \frac{\text{UNSAT}_a(G)|E|}{2}$, then $|S| \geq \frac{\text{UNSAT}_a(G)|E|}{2}$. Pick any v and let $S^v = [v] \cap S$ be the set of vertices not agreeing with the plurality assignment, clearly $|S^v| \leq \frac{|[v]|}{2}$. Define

  $$S^v_\sigma := \{(v, e) \in S^v | a(v, e) = \sigma\}$$

  We have that $|S^v_\sigma| \leq \frac{|[v]|}{2}$, so by the expansion property of the graph on $[v]$, $E(S^v_\sigma, [v] \setminus S^v_\sigma) \geq h_0|S^v_\sigma|$. All of the edges incident to S^v_σ within $[v]$ have equality constraints that are rejected by a'. Ranging over every $\sigma \in \Sigma$ the sets S^v_σ are pairwise disjoint and span S^v. So, there are

  $$\frac{1}{2} \sum_{\sigma \in \Sigma \setminus a(v)} E(S^v_\sigma, [v] \setminus S^v_\sigma) \geq \frac{h_0}{2}|S^v|$$

  edges rejected by a' in the cloud $[v]$. So in total, there are at least

  $$\frac{h_0}{2} \sum_{v \in V} |S \cap [v]| = \frac{h_0}{2}|S| \geq \frac{h_0 \text{UNSAT}_a(G)|E|}{4} \geq \frac{h_0}{4d}\text{UNSAT}_a(G)|E'|$$

  edges rejected by a'. In other words $\text{UNSAT}_{a'}(G') \geq \frac{h_0}{4d}\text{UNSAT}_a(G)$.

We conclude that $c \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G')$, with $c = \min\left(\frac{1}{2d}, \frac{h_0}{4d}\right)$. □

Next, we add a self-loop to each vertex and add the edges of an expander graph with the same vertex set. This increases the expansion rate of the graph. We add no constraints to the new expander edges and the self-loops.

Definition 4.11 (Preprocessing step 2)**.** Let $G = ((V, E), \Sigma, \mathcal{C})$ be a constraint graph, we define $G' := prep_2(G) := ((V, E'), \Sigma, \mathcal{C}')$ as follows.

- Let $H = (V, E_1)$ be a d_0'-regular expander graph with $\lambda(H) < \lambda_0 < d_0'$ as guaranteed by lemma 4.14 and let $E_2 := \{(v, v) | v \in V\}$ be the set of self-loops on each vertex. Finally, we define $E' := E \cup E_1 \cup E_2$.
- For each $e' \in E$ we define the constraints as follows.
  - If $e' \in E$, $c(e') = c(e)$.
  - If $e' \in E_1 \cup E_2$, $c(e') = \Sigma \times \Sigma$, the null constraint that is always satisfied.

Dinur shows that the new graph has the desired properties.

Lemma 4.16. *There are constants $d_0' > \lambda_0 > 0$ such that for any d-regular constraint graph G, the constraint graph $G' := prep_2(G)$ has the following properties:*

- *G' is a $(d + d_0' + 1)$-regular constraint graph with a self-loop on every vertex.*
- *$\lambda(G') \leq d + \lambda_0 + 1 < deg(G')$.*
- *$Size(G') = O(Size(G))$.*
- *If some assignment $a : V \to \Sigma$ satisfies G, then a satisfies G'.*
- *$\frac{d}{d+d_0'+1} UNSAT(G) \leq UNSAT(G') \leq UNSAT(G)$.*

Proof. Clearly, G' is a $(d + d_0' + 1)$-regular constraint graph with a self-loop on every vertex.

Since $\lambda_0 < d_0'$, we have that $d + \lambda_0 + 1 < deg(G)$. We show that $\lambda(G') \leq d + \lambda_0 + 1$ using lemma 4.12.

$$\begin{aligned}
\lambda(G') &= \max_{\|x\|=1, \langle x, \vec{1}\rangle = 0} |\langle x, A_{G'} x\rangle| \\
&\leq \max_{\|x\|=1, \langle x, \vec{1}\rangle = 0} |\langle x, A_G x\rangle| + \max_{\|x\|=1, \langle x, \vec{1}\rangle = 0} |\langle x, A_X x\rangle| + \max_{\|x\|=1, \langle x, \vec{1}\rangle = 0} |\langle x, Ix\rangle| \\
&= \lambda(G) + \lambda(X) + 1 \\
&\leq d + \lambda_0 + 1
\end{aligned}$$

We also have that $Size(G') = |E'| + |V'| = \frac{d}{d+d_0'+1}|E| + |V| = O(Size(G))$.

Let $a : V \to \Sigma$ be an assignment and let $F := F_a \subseteq E$ and $F' := F_a' \subseteq E'$ be the sets of edges not satisfied by a. Noting that $|F| = |F'|$, the following equations hold.

$$\text{UNSAT}_a(G') = \frac{|F'|}{|E'|} = \frac{|E|}{|E'|}\frac{|F'|}{|E|} = \frac{d}{d + d_0' + 1}\frac{|F|}{|E|} = \frac{d}{d + d_0' + 1}\text{UNSAT}_a(G)$$

So, we have that $c_2 \cdot \text{UNSAT}(G) = \text{UNSAT}(G')$, with $c_2 = \frac{d}{d+d_0'+1}$. If a satisfies G, then $|F| = |F'| = 0$, so, a satisfies G'.

Now, let a be an assignment such that $\text{UNSAT}_a(G) = \text{UNSAT}(G)$, we have the following inequality:

$$\text{UNSAT}(G) = \text{UNSAT}_a(G) \geq \frac{d}{d + d_0' + 1}\text{UNSAT}_a(G) = \text{UNSAT}_a(G') \geq \text{UNSAT}(G')$$

Let a' be an assignment such that $\text{UNSAT}_{a'}(G') = \text{UNSAT}(G')$, we have the following inequality:

$$\text{UNSAT}(G') = \text{UNSAT}_{a'}(G') = \frac{d}{d + d'_0 + 1}\text{UNSAT}_{a'}(G) \geq \frac{d}{d + d'_0 + 1}\text{UNSAT}(G) \qquad \square$$

Combining the two preprocessing steps, we get the following proposition for $prep := prep_2(prep_1(G))$.

Proposition 4.17. *There exists constants $0 < \lambda < d$ and β_1 such that any constraint graph G can be transformed into a constraint graph $G' := prep(G) := prep_2(prep_1(G))$ such that*

- *G' is d-regular with a self-loop on each vertex.*
- *$\lambda(G') \leq \lambda < d$.*
- *G' has the same alphabet as G.*
- *$size(G') = O(size(G))$*
- *$\beta_1 \cdot UNSAT(G) \leq UNSAT(G') \leq UNSAT(G)$.*

Proof. This proposition follows directly from lemma 4.15 and 4.16. $\square$

4.2.5 Amplification

The properties of the preprocessed graph allow us to apply the amplification step. The intuition for this step is as follows. Given a constraint graph $G = ((V, E), \Sigma, \mathcal{C})$, instead of each constraint checking one edge at once, we would like to check multiple, say t, edges. Then, the probability of hitting a "bad" edge at least once is roughly multiplied by t. In this section we omit most technical details of this steps and only cover the construction of the amplified graph. For details, we refer the reader to [9].

We create a new graph where each edge represents a *t-step walk* through G and a constraint checks whether any of the t edges in this walk is "bad". This process increases the unsat value of a preprocessed constraint graph by approximately a factor $\sqrt{t}$.

Definition 4.12. Let $G = (V, E)$ be a graph. We define a *t-step walk* on G to be a sequence of nodes $(v_0, v_1, \ldots, v_t)$. We call v_0 the *start* and v_t the *end* of the t-step walk and say that this is walk goes from v_0 to v_t.

Consider the following probabilistic algorithm for randomly selecting a t-step walk on G.

1. First, pick a vertex $v = v_0$ uniformly at random and let $i = 0$.
2. Pick an edge (u, v_i) connected to v_i uniformly at random and let v_{i+1}.
3. Repeat step 2 until $i = t$.

Note that a double edge has twice the probability to get chosen in step 2. This probability distribution naturally defines a weighted graph, where the weight of an edge (u, v) is the probability that a random walk starting from u ends in v. From a weighted graph we can construct a multigraph without weights by multiplying each weight with the total number of walks.

Definition 4.13. Let $G = ((V, E), \Sigma, \mathcal{C})$ be a d-regular constraint graph. We define the constraint graph $G^t := ((V, E'), \Sigma^{d^{\lceil \frac{t}{2} \rceil}}, \mathcal{C}')$ as follows.

- The vertex set remains the same.
- Let $u \in V$ and $v \in V$ be two vertices. Let p be the probability that a random walk that starts at u ends in v and let w be the total number of random walks on G. Add $p \cdot w$ edges between u and v.

- For every node $v \in V$, let

$$\Gamma(v) := \{u \in V | (v = u_0, u_1, \ldots, u_{\lceil \frac{t}{2} \rceil} = u) \text{ is a t-step walk in } G\}$$

  Since $|\Gamma(v)| \leq d^{\lceil \frac{t}{2} \rceil}$, we can interpret a value $a \in \Sigma^{d^{\lceil \frac{t}{2} \rceil}}$ for v as an assignment $a : \Gamma(v) \to \Sigma$. We write $a_{v'}$ for the value a assigns to some $v' \in \Gamma(v)$.

- We describe C' by stating which values satisfy a constraint on an edge. Let $e' = (u, v) \in E'$ and $a, b \in \Sigma^{d^{\lceil \frac{t}{2} \rceil}}$. Now (a, b) satisfies $c(e')$ if there is an assignment $\sigma : \Gamma(u) \cup \Gamma(v) \to \Sigma$ such that:
  - For each $e \in E \cap (\Gamma(u) \times \Gamma(v))$, $c(e)$ is satisfied by σ.
  - For each $u' \in \Gamma(u)$ and $v' \in \Gamma(v)$, it holds that $\sigma(u') = a_{u'}$ and $\sigma(v') = b_{v'}$.

The amplifying step with parameter t increases the unsat value of a graph with a factor of $O(\sqrt{t})$.

Proposition 4.18. *Let $0 < \lambda < d$ be constants and let Σ be an alphabet of constant size. There exists a constant $\beta_2 = \beta_2(\lambda, d, |\Sigma|)$ such that for every $t \in \mathbb{N}$, given a d-regular constraint graph $G = ((V, E), \Sigma, \mathcal{C})$ with a self-loop on every vertex and $\lambda(G) \leq \lambda$ it holds that $Size(G^t) = O(Size(G))$ and*

$$UNSAT(G^t) \geq \beta_2 \sqrt{t} \min(UNSAT(G), \frac{1}{t}). \tag{2}$$

Lastly, we have that if $UNSAT(G) = 0$, then $UNSAT(G^t) = 0$.

Proof. The number of edges increases by a factor of at most $d^{\lceil \frac{t}{2} \rceil}$ and the vertex set remains the same. So, the size of the constraint graph increases by only a constant factor.

We use the following technical lemma to prove that the amplification step increases the unsat value of a graph by this factor.

Lemma 4.19. *Let $a' : V \to \Sigma^{d^{\lceil \frac{t}{2} \rceil}}$ be an assignment, we define an assignment $a : V \to \Sigma$ as follows. For every v, let $a(v)$ be the value for v that appears the most in the assignment a':*

$$a(v) := \max \arg_{\sigma \in \Sigma} \{\mathbb{P}[A \text{ random } \lceil \frac{t}{2} \rceil \text{-step walk from } v \text{ has a vertex } u \text{ for which } a'(u)_v = \sigma \text{ as endpoint}]\}.$$

Then,

$$UNSAT_{a'}(G^t) \geq \beta_2 \sqrt{t} \min(UNSAT_a(G), \frac{1}{t}).$$

Proof. The proof of this lemma revolves around analyzing the probability that the "middle section" of a random walk contains at least one edge that is not satisfied by the popular opinion. We refer to the proof of Dinur for details of this proof [9]. □

From this lemma, we can prove equation (2). Let a' be an assignment such that $\text{UNSAT}_{a'}(G^t) = \text{UNSAT}(G^t)$. We have that

$$\text{UNSAT}(G^t) = \text{UNSAT}_{a'}(G^t) \geq \beta_2 \sqrt{t} (\text{UNSAT}_a(G), \frac{1}{t}) \geq \beta_2 \sqrt{t} (\text{UNSAT}(G), \frac{1}{t}).$$

In the case that $\text{UNSAT}(G) = 0$, we can assign each $v \in V$ a value $a(v)$ such that all constraints in G are satisfied. Looking at G^t, we assign each $v \in V$ a value $a'(v) \in \Sigma^{d^{\lceil \frac{t}{2} \rceil}}$ such that for each $u \in \Gamma(v)$, $a'_u = a(u)$. This assignment satisfies all constraints of G^t and thus $\text{UNSAT}(G^t) = \text{UNSAT}_{a'}(G^t) = 0$. □

4.2.6 Alphabet reduction

The amplification steps results in a graph with an increased gap size. However, the size of the alphabet is increased by a large factor. Repeating amplification would result in the alphabet size increasing rapidly. To resolve this issue, we perform a standard alphabet reduction step. We reduce the alphabet to one of constant size, while decreasing the unsat value and increasing the size of the graph by only a constant factor. We use the methods from Dinur [9].

The alphabet reduction step makes use of an *assignment tester*. First, we define an assignment tester and show how it can be used to reduce that alphabet of a constraint graph G. Later in this section, we explicitly construct one.

An assignment tester takes as input a boolean circuit. A boolean circuit formalizes the concept of a computer circuit. We refer to [21, Chapter 9] for a formal definition.

Given a boolean circuit Φ over variable set $X = \{x_1, x_2, \ldots, x_n\}$, we identify an assignment to the variables, $a : X \to \{0,1\}$, with the binary string $a \in \{0,1\}^n$ by letting the k-th bit of the string be equal to the value $a(x_k)$. We write $SAT(\Phi)$ for the set of assignments that satisfy Φ.

Definition 4.14. Let $a, b \in \{0,1\}^n$ be two strings of equal length, the *distance* between a and b, denoted $dist(a,b)$, is the number of positions where the corresponding values are different between the two strings. The *relative distance* is defined as

$$rdist(a,b) := \frac{1}{n} dist(a,b)$$

The relative distance between a and a set of length n strings $C \subset \{0,1\}^n$ is defined as

$$rdist(a,C) := \min_{c \in C} rdist(a,c)$$

We encode the alphabet Σ into binary using an encoding $e : \Sigma \to \{0,1\}^l$. We say that an encoding has *relative distance* ρ if for each $\sigma_1, \sigma_2 \in \Sigma$ with $\sigma_1 \neq \sigma_2$ we have that

$$rdist(e(\sigma_1), e(\sigma_2)) \geq \rho$$

We require our encoding to have strictly positive relative distance. We call such an encoding an error correcting code. We also require our encoding to be of linear dimension, meaning that $l = O(\log_2 |\Sigma|)$.

Definition 4.15. Let Σ_0 be a finite alphabet and let $\epsilon > 0$ be a constant. An *assignment tester* with rejection probability ϵ is a Turing machine $\mathcal{P}$ that takes as input a boolean circuit Φ over variable set X and outputs a constraint graph $G = ((V,E), \Sigma_0, \mathcal{C})$ with $X \subset V$. Define $V' = V \setminus X$. For any assignment $a : X \to \{0,1\}$ the following holds.

- (Completeness) If $a \in SAT(\Phi)$, there exists $b : V' \to \Sigma_0$, such that $\text{UNSAT}_{a \cup b}(G) = 0$.
- (Soundness) If $a \notin SAT(\Phi)$, then for any $b : V' \to \Sigma_0$, it holds that $\text{UNSAT}_{a \cup b}(G) \geq \epsilon \cdot rdist(a, SAT(\Phi))$.

Proposition 4.20 ([9])**.** *There exists an assignment tester with alphabet $\Sigma_0 = \{0,1\}^3$ and fixed rejection probability $\epsilon > 0$.*

Later in this section we prove this statement by explicitly constructing an assignment tester with these properties. Using this assignment tester, we reduce the alphabet size of a constraint graph.

Definition 4.16. Let $G = ((V,E), \Sigma, \mathcal{C})$ be a constraint graph and $\mathcal{P}$ be an assignment tester. Let $l := \lceil \log |\Sigma| \rceil$ and let $enc : \Sigma \to \{0,1\}^l$ be an arbitrary but fixed encoding with linear dimension and relative distance $\rho > 0$. Let $\Sigma_0 = \{0,1\}^3$. We define the constraint graph $G \circ \mathcal{P} = ((V', E'), \Sigma_0, \mathcal{C}')$ as follows.

- First, we convert each constraint $c(e) \in \mathcal{C}$ into a circuit $\tilde{c}(e)$. For each variable $v \in V$, let $[v] := \{v_1, v_2, \ldots, v_l\}$ be a set of l boolean variables. For each edge $e = (u,v)$, $\tilde{c}(e)$ will be a circuit on the $2l$ variables $[u] \cup [v]$. Given an assignment $a : [u] \cup [v] \to \{0,1\}^{2l}$, $\tilde{c}(e)$ outputs 1 iif there exists some $\sigma_1, \sigma_2 \in \Sigma$ such that $enc(\sigma_1) = [u]$, $enc(\sigma_2) = [v]$ and $(\sigma_1, \sigma_2) \in c(e)$.

- Let $((V_e, E_e), \Sigma_0, \mathcal{C}_e) = G_e := \mathcal{P}(\tilde{c}(e))$ be the constraint graph resulting from running $\mathcal{P}$ on $\tilde{c}(e)$. We finally define $G \circ \mathcal{P} = ((V', E'), \Sigma_0, \mathcal{C}')$ with
  - $V' = \bigcup_{e \in E} V_e$
  - $E' = \bigcup_{e \in E} E_e$
  - $\mathcal{C}' = \bigcup_{e \in E} \mathcal{C}_e$

Proposition 4.21. *There exists a constant $\beta_3 > 0$ that depends only on $\mathcal{P}$ and a constant c that depends only on $\mathcal{P}$ and $|\Sigma|$, such that the following holds. Given any constraint graph $G = ((V, E), \Sigma, \mathcal{C})$, the constraint graph $G' = G \circ \mathcal{P}$ from definition 4.16 can be computed in time linear in $size(G)$ such that $size(G') = c \cdot size(G)$ and*

$$\beta_3 \cdot UNSAT(G) \leq UNSAT(G') \leq UNSAT(G)$$

Proof. First, we prove that $\text{UNSAT}(G') \leq \text{UNSAT}(G)$. Let $a : V \to \Sigma$ be an assignment such that $\text{UNSAT}_a(G) = \text{UNSAT}(G)$. Given a, we construct an assignment $a' : V' \to \Sigma_0$ and show that $\text{UNSAT}_{a'}(G') \leq \text{UNSAT}_a(G)$. For each $v \in V$, set $a'([v]) = enc(a(v)) \in \{0,1\}^l$, where $a'([v])$ is the concatenation of $a'(v_1), a'(v_2), \ldots, a'(v_l)$. The value of a' for the variables $\{v_1, \ldots, v_l\}$ corresponding to v is the encoding of $a(v)$.

The assignment a' is now defined for $\bigcup_{v \in V}[v]$, so it remains to define values for $V_e \setminus [u] \cup [v]$, for each $e = (u, v) \in E$. We separate two cases.

- If $c(e)$ is satisfied by a, then by definition of $\tilde{c}(e)$, this circuit is satisfied by $a'([u] \cup [v])$. Now, by the completeness property of the assignment tester, there exists an assignment $b' : V_e \setminus [u] \cup [v] \to \Sigma_0$ such that $\text{UNSAT}_{a' \cup b'}(G_e) = 0$. We now define $a'(V_e \setminus [u] \cup [v]) := b'(V_e \setminus [u] \cup [v])$.
- If $c(e)$ is not satisfied, we assign values to $V_e \setminus [u] \cup [v]$ arbitrarily.

Since each E_e has the same cardinality, say equal to k, we have that

$$\text{UNSAT}_{a'}(G') = \frac{|F_{a'}|}{|E'|} = \frac{1}{|E||k|} \sum_{e \in E} |F_{a',e}| \tag{3}$$

$$= \frac{1}{|E|} \sum_{e \in E} \frac{|F_{a',e}|}{|k|} = \frac{1}{|E|} \sum_{e \in E} \frac{|F_{a',e}|}{|E_e|} \tag{4}$$

$$= \frac{1}{|E|} \sum_{e \in E} \text{UNSAT}_{a'|_{V_e}}(G_e) \tag{5}$$

Where the second equality holds because the set of E_e partition E' and thus the violated edges in E' are partitioned by the violated edges of E_e.

$$\begin{aligned}
\text{UNSAT}_{a'}(G') &= \frac{1}{|E|} \sum_{e \in E} \text{UNSAT}_{a'|_{V_e}}(G_e) \\
&= \frac{1}{|E|} \sum_{e \in F_a} \text{UNSAT}_{a'|_{V_e}}(G_e) \\
&\leq \frac{1}{|E|} \sum_{e \in F_a} 1 \\
&= \frac{|F_a|}{|E|} = \text{UNSAT}_a(G)
\end{aligned}$$

Where the second equality holds because when an edge $e \in E$ is satisfied by a, the resulting graph G_e is satisfied by a', by construction of a'. We conclude that

$$\text{UNSAT}(G') \leq \text{UNSAT}_{a'}(G') \leq \text{UNSAT}_a(G) = \text{UNSAT}(G).$$

Next, we show that for some $\beta_3 > 0$, it holds that $\beta_3 \cdot \text{UNSAT}(G) \leq \text{UNSAT}(G')$. Let $a' : V' \to \Sigma_0$ be an assignment such that $\text{UNSAT}_{a'}(G') = \text{UNSAT}(G')$. We construct the assignment $a : V \to \Sigma$ by for each $v \in V$ defining $a(v) := \arg\min_{\sigma \in \Sigma} [rdist(e(\sigma), a'([v]))]$, interpreting the values $000, 111 \in \Sigma_0$ as the values 0 and 1 respectively. In other words, we let $a(v)$ be the value whose encoding is closest to $a'([v])$.

Let $e = (u, v) \in F_a$ be an edge whose constraint is not satisfied by a and consider the relative distance of $a'([u] \cup [v])$ and $SAT(\tilde{c}(e))$. First, we note that $a'([u] \cup [v]) \notin SAT(\tilde{c}(e))$. The opposite would imply that $(a(u), a(v)) \in c$ and $(u, v) \notin F_a$.

We analyze the minimum number of bits that $a'([u] \cup [v])$ differs from any satisfying assignment of $\tilde{c}(e)$. Without loss of generality we can assume that $a'([v])$ is the encoding of some $\sigma' \in \Sigma$. Consider altering the value of $a'([u])$ to a value that is an encoding of some $\sigma \in \Sigma$ such that $(\sigma, \sigma') \in c$. To achieve this, we need to change at least $\frac{\rho}{2}$ bits. Assume for the argument of contradiction that we could change $a'([u])$ to a value encoding σ by altering no more than $\frac{\rho}{2}$ bits, then $rdist(enc(\sigma), a'([u])) < rdist(enc(a(u)), a'([u]))$, which is in contradiction with the definition of $a(u)$. So, it holds that

$$rdist(a'([u] \cup [v]), SAT(\tilde{c}(e))) \geq \frac{\rho}{4}$$

By the soundness of $\mathcal{P}$, we have that $\text{UNSAT}_{a'|_{V_e}}(G_e) \geq \epsilon\frac{\rho}{4}$. We define $\beta_3 := \frac{\epsilon\rho}{4}$. Now, by the assumption that $|E_e|$ is equal for each $e \in E$ and using equation (5), we have that

$$\begin{aligned}
\text{UNSAT}(G') &= \text{UNSAT}_{a'}(G') \\
&= \frac{1}{|E|} \sum_{e \in E} \text{UNSAT}_{a'|_{V_e}}(G_e) \\
&\geq \frac{1}{|E|} \sum_{e \in F_a} \text{UNSAT}_{a'|_{V_e}}(G_e) \\
&\geq \frac{1}{|E|} \sum_{e \in F_a} \beta_3 \\
&= \beta_3 \frac{|F_a|}{|E|} = \beta_3 \text{UNSAT}_a(G) \geq \text{UNSAT}(G).
\end{aligned}$$

It remains to show that this reduction runs in linear time. Note that in proposition 4.20 we do make a claim about the run-time of the assignment tester. We show that still the assignment tester runs in time linear in $size(G)$.

Let $f : \mathbb{N} \to \mathbb{N}$ be the function that, given the size n of a boolean circuit, outputs the number of operations performed by the Turing Machine $\mathcal{P}$. Consider the size of a boolean circuit as constructed in definition 4.16. This circuit can be constructed in time $O(2^l)$, which is still constant as a function of $size(G)$. The maximum size of this circuit is a function of only l and $|\Sigma|$. Since l itself is a function of $|\Sigma|$ and $|\Sigma|$ stays bounded, we conclude that the size of any circuit stays bounded by some constant k. Given a circuit with size bounded by k, the run time of $\mathcal{P}$ is at most $f(k)$, which is again a constant. We construct $|E| \leq size(G)$ circuits. So, our computation is done in time linear in $size(G)$. A direct consequence is that the size of G' is also linear in $size(G)$, so $size(G') = c \cdot size(G)$ for some constant c. □

Lemma 4.22. *The reduction $G \circ \mathcal{P}$ has a linear time witness transformation.*

Proof. Proposition 4.21 tells us that we can indeed transform a witness for G into a witness for $G \circ \mathcal{P}$. Given an edge $e = (u, v)$ and given an assignment $a' : [u] \cup [v]$ satisfying $\tilde{c}(e)$, we can construct the assignment $b' : V_e \setminus [u] \cup [v]$ such that $\text{UNSAT}_{a' \cup b'}(G_e) = 0$ in time $O(f(l))$. Here $f : \mathbb{N} \to \mathbb{N}$ is a function depending only on l. We can in time $O(2^l)$ construct $a'_1([v]) = enc(a(v))$ and in similar time construct $a'_2([v]) = enc(a(v))$.

So, we can construct an assignment for V_e in time that is dependent only on l. By the same reasoning from proposition 4.21, this value is a constant, since $|\Sigma|$ and thus l is bounded. We construct an $|E|$ number of such assignments. This can be done in linear time. □

Constructing an assignment tester
Dinur explicitly constructs an assignment tester with alphabet $\Sigma_0 = \{0,1\}^3$ and error probability $\epsilon > 0$. To achieve this, Dinur constructs two PCP verifiers and a constraint graph that mimics the behavior of these verifiers in a deterministic manner.

We introduce the *long code* of an assignment. The long code is a redundant encoding of a binary string. This redundancy is used to test whether an assignment is valid by checking a low number of bits in the long code.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of binary-valued variables. We denote $\{0,1\}^X$ for the set of all possible assignments to X, that is, all binary strings of size $|X|$, or equivalently, all functions $a : X \to \{0,1\}$. Let $\mathcal{F}_X := \{f : \{0,1\}^X \to \{0,1\}\}$ be the set of functions from $\{0,1\}^X$ to $\{0,1\}$.

Definition 4.17. The *long code* of assignment $a \in \{0,1\}^X$ is the function $A_a : \mathcal{F}_X \to \{0,1\}$ defined by $f \mapsto f(a)$.

The verifier T_1 is given as input a boolean circuit Φ over variable set X and has oracle access to a function $A : \mathcal{F}_X \to \{0,1\}$. It tests whether A is the long code of some assignment satisfying Φ. The verifier T_1 is defined by the following algorithm.

Verifier T_1

Input: A boolean circuit Φ.
Proof: A function $A : \mathcal{F}_X \to \{0,1\}$.

1. Pick $f, g \in \mathcal{F}_X$ uniformly at random.
2. Define function $\mu \in \mathcal{F}_X$, by for each $y \in \{0,1\}^X$ setting
   - $\mu(y) = 1$, if $f(y) = 0$.
   - $\mu(y) = 0$ with probability $\frac{99}{100}$, $\mu(y) = 1$ with probability $\frac{1}{100}$, if $f(y) = 1$.
3. Define function $h \in \mathcal{F}_X$ as follows. For each $y \in \{0,1\}^X$, let $h(y) := g(y) \oplus \mu(y)$, where $\oplus$ denotes the XOR operation.
4. Accept unless $A(f) = A(g) = A(h) = 1$.

We prove the following lemmas on the completeness and soundness of T_1.

Lemma 4.23. *If $SAT(\Phi) \neq \emptyset$, then there exists some $A : \mathcal{F}_X \to \{0,1\}$ such that $\mathbb{P}[T_1^A = 1] = 1$.*

Proof. Let $a \in SAT(\Phi)$ be any satisfying assignment, let $A = A_a$ and let $f, g \in \mathcal{F}_X$ be functions. If $A(f) = f(a) = 1$, then $\mu(a) = 1$ and $A(h) = h(a) = g(a) \oplus \mu(a) = \overline{g(a)} = \overline{A(g)}$. This means that $A(g) \neq A(h)$ and T_1 accepts. If $A(f) = 0$, T_1 also accepts. So, the probability that T_1 accepts is 1. □

Lemma 4.24 ([9])**.** *Let $LONGSAT(\Phi) := \{A_a : \mathcal{F}_X \to \{0,1\} \mid a \in SAT(\Phi)\}$ be the set of long codes of assignments satisfying Φ. There exists a constant $c > 0$ such that for every $\delta \in [0,1]$, if for the function $A : \mathcal{F}_X \to \{0,1\}$ it holds that $rdist(A, LONGSAT(\Phi)) \geq \delta$, then $\mathbb{P}[T_1^A = 0] \geq c \cdot \delta$.*

The second verifier, T_2, also takes a boolean circuit Φ over variable set X as input. However, T_2 has oracle access to both a function $A : \mathcal{F}_X \to \{0,1\}$ and a function $a : X \to \{0,1\}$. The verifier T_2 is defined by the following algorithm.

Verifier T_2

Input: A boolean circuit Φ.
Proof: A function $A : \mathcal{F}_X \to \{0,1\}$ and an assignment $a : X \to \{0,1\}$.

1. Pick $x_i \in X$ and $f \in \mathcal{F}_X$ uniformly at random.

2. Accept if $a(x_i) = A(f) \oplus A(f + e_i)$, where $e_i : \{0,1\}^X \to \{0,1\}$ is the function that given an assignment $b \in \{0,1\}^X$, outputs $b(x_i)$.

We prove two statements on the completeness and soundness of T_2.

Lemma 4.25. *If $SAT(\Phi) \neq \emptyset$, then there exists some $A : \mathcal{F}_X \to \{0,1\}$ and $a : X \to \{0,1\}$ such that $\mathbb{P}[T_2^{A,a} = 1] = 1$.*

Proof. Let $a \in SAT(\Phi)$ be any satisfying assignment and let $A = A_a$. Now

$$A(f) \oplus A(f + e_i) = f(a) \oplus (f \oplus e_i)(a) = f(a) \oplus f(a) \oplus e_i(a) = e_i(a) = a(x_i).$$

□

Lemma 4.26. *If for assignment $a : X \to \{0,1\}$ it holds that $rdist(a, SAT(\Phi)) \geq \delta$ for some $\delta > 0$ and for $A : \mathcal{F}_X \to \{0,1\}$ it holds that $rdist(A, LONGSAT(\Phi)) \leq \frac{\delta}{2}$, then $\mathbb{P}[T_2^{A,a} = 0] \geq \frac{1}{2}\delta$.*

Proof. Let $a' \in SAT(\Phi)$ be an assignment such that $rdist(A, A_{a'}) \leq \frac{\delta}{2}$. We compare a and a'. Since $a' \in SAT(\Phi)$ it holds that

$$\mathbb{P}_i[a(x_i) \neq a'(x_i)] = rdist(a, a') \geq rdist(a, SAT(\Phi)) = \delta$$

For every i we have

$$\begin{aligned}\mathbb{P}_{f \in \mathcal{F}_X}[(A(f) \oplus A(f + e_i) = a'(x_i)] &\geq \mathbb{P}_{f \in \mathcal{F}_X}[(A(f) = f(a') \text{ and } A(f + e_i) = (f \oplus e_i)(a')] \\ &= 1 - \mathbb{P}_{f \in \mathcal{F}_X}[(A(f) \neq f(a') \text{ or } A(f + e_i) \neq (f \oplus e_i)(a')] \\ &\geq 1 - (\frac{\delta}{2} + \frac{\delta}{2}) = 1 - \delta\end{aligned}$$

When $a(x_i) \neq a'(x_i)$ and $A(f) \oplus A(f + e_i) = a'(x_i)$, T_2 rejects. These two events happen with probability at least δ and at least $1 - \delta$ respectively, so T_2 rejects with probability at least $(1 - \delta)\delta \geq \frac{\delta}{2}$. □

We combine T_1 and T_2 to construct the verifier T. This verifier also takes a boolean circuit Φ over variable set X as input and has oracle access to both a function $A : \mathcal{F}_X \to \{0,1\}$ and a function $a : X \to \{0,1\}$. T runs either T_1 or T_2 with equal probability.

Verifier T

Input: A boolean circuit Φ.
Proof: A function $A : \mathcal{F}_X \to \{0,1\}$ and an assignment $a : X \to \{0,1\}$.

1. With probability $\frac{1}{2}$, run T_1 and accept if $T_1^A(\Phi) = 1$.

2. Else, with probability $\frac{1}{2}$, run T_2 and accept if $T_2^{A,a}(\Phi) = 1$.

We use this verifier T to explicitly construct an assignment tester.

Proof of proposition 4.20. We describe a Turing machine that takes as input a boolean circuit Φ and returns a constraint graph G. We then prove that this Turing machine is an assignment tester with alphabet $\Sigma_0 = \{0,1\}^3$ and constant rejection probability $\epsilon > 0$. Note that we do not make any claims about the runtime of the Turing machine.

We consider applying the test T to Φ. We introduce a new set of variables Z containing a new variable z_r for each roll of the random coins in T. These variables take values in $\{0,1\}^3$, representing the values of the 3 bits in the proof queried by T. Using this test, we construct the following constraint graph $G = ((V, E), \Sigma_0, \mathcal{C})$:

- The vertices are the set $V := X \cup \mathcal{F}_X \cup Z$.

- For each $z_r \in Z$, let $y_{r1}, y_{r2}, y_{r3} \in X \cup \mathcal{F}_X$ be the variables queried by T on the string r. We define the set of edges to be $E := \bigcup_r (z_r, y_{r1}) \cup (z_r, y_{r2}) \cup (z_r, y_{r3})$.

- The alphabet is $\Sigma_0 = \{0,1\}^3$.
- For each edge $e = (z_r, y_{ri}) \in E$, we have the constraint $c(e)$ that checks whether the value given to z_r would cause the test T to accept and that checks whether the bit in z_r corresponding to y_{ri} has the same value as y_{ri}. Here $000, 111 \in \{0,1\}^3$ represent 0 and 1 respectively and a constraint immediately rejects if the value of y_{ri} is not 000 or 111.

Clearly $X \subset V$. Define $V' = V \setminus X = \mathcal{F}_X \cup Z$. We show that the completeness of the assignment tester is 1 and the soundness is at most $1 - \epsilon \cdot rdist(a, SAT(\Phi))$.

- (Completeness) Let $a \in SAT(\Phi)$ be a satisfying assignment. We define $b : V' \to \Sigma_0$ to be the following assignment. Let $A := A_a$ and assign to each $z_r \in Z$ the value that would make T accept on randomness r and is consistent with the assignment of X and $\mathcal{F}_X$. By the completeness lemmas 4.23 and 4.25 of T_1 and T_2, this means that $\text{UNSAT}_{a \cup b}(G) = 0$.
- (Soundness) Let $a \notin SAT(\Phi)$ with $rdist(a, SAT(\Phi)) \geq \delta$. Let $b : V' \to \Sigma_0$ be an assignment, b defines the function $A : \mathcal{F}_X \to \{0,1\}$.

  Let r be a random string causing T to reject on the proof A, a. We consider two cases:

  - The value $b(z_r)$ is consistent with A and a. In this case 3 of the 3 constraints associated with r are rejected.
  - The value $b(z_r)$ is inconsistent with at least one of the 3 values of A and a queried by T^r. In this case at least 1 of the 3 constraints associated with r are rejected.

  Any r is rejected with probability $\Omega(\delta)$, so the probability of an edge rejecting is at least $\frac{\Omega(\delta)}{3}$. So $\text{UNSAT}_{a \cup b}(G) \geq \epsilon \cdot rdist(a, SAT(\Phi))$. □

4.2.7 Full reduction

We have presented all steps necessary for the final reduction that increases the gap of a constraint graph to a gap of constant size.

Let G be a constraint graph. We define

$$step(G) := (prep(G))^t \circ \mathcal{P}$$

Here, *prep* is the operation from proposition 4.17, the power t is the operation from definition 4.12 and $\circ\mathcal{P}$ the operation from definition 4.16. Dinur shows that the operation *step* doubles the unsat value of a constraint graph, unless that value is larger than some constant.

Proposition 4.27. *For any finite alphabet Σ, there exists constants C and $0 < \gamma < 1$ such that, given a constraint graph $G = ((V, E), \Sigma, \mathcal{C})$, one can in polynomial time construct the graph $G' = step(G) = ((V', E'), \Sigma_0, \mathcal{C})$, where $\Sigma_0 = \{0,1\}^3$, such that*

- *$Size(G') \leq C \cdot Size(G)$.*
- *If $UNSAT(G) = 0$, then $UNSAT(G') = 0$.*
- *If $UNSAT(G) > 0$, then $UNSAT(G') \geq \min(2 \cdot UNSAT(G), \gamma)$.*

Proof. We describe the steps taken in constructing $G' := (prep(G))^t \circ \mathcal{P}$.

- Let $H_1 := prep(G)$. By proposition 4.17, H_1 is d-regular and has a self-loop on each vertex. Further, we have that $\lambda(H_1) \leq \lambda$ for some global constant λ and $\beta_1 \cdot \text{UNSAT}(G) \leq \text{UNSAT}(H_1) \leq \text{UNSAT}(G)$ for some global constant $\beta_1 > 0$.
- Let $H_2 := H_1^t$. By proposition 4.18 it holds that $\text{UNSAT}(H_2) \geq \beta_2 \sqrt{t} \min\left(\text{UNSAT}(H_1), \frac{1}{t}\right)$ for some global constant $\beta_2 > 0$.

- Let $G' := H_2 \circ \mathcal{P}$. By proposition 4.21 the alphabet of G' is Σ_0 and we have that $\beta_3 \cdot \text{UNSAT}(H_2) \leq \text{UNSAT}(G') \leq \text{UNSAT}(H_2)$ for some global constant $\beta_3 > 0$.

The size of the graph only increases linearly for each step, so $size(G') = O(size(G))$.

If $\text{UNSAT}(G) = 0$, then clearly $\text{UNSAT}(H_1) = 0$, $\text{UNSAT}(H_2) = 0$ and $\text{UNSAT}(G') = 0$.

Note that we have not fixed a value for t yet. We choose a value such that the reduction results in the doubling of the unsat value.

Let $t := \lceil (\frac{2}{\beta_1\beta_2\beta_3})^2 \rceil$ and define $\gamma := \frac{\beta_2\beta_3}{\sqrt{t}}$. If $\text{UNSAT}(G) > 0$, it holds that

$$\begin{aligned}
\text{UNSAT}(G') &\geq \beta_3 \text{UNSAT}(H_2) \\
&\geq \beta_3\beta_2\sqrt{t}\min(\text{UNSAT}(H_1), \frac{1}{t}) \\
&\geq \beta_3\beta_2\sqrt{t}\min(\beta_1\text{UNSAT}(G), \frac{1}{t}) \\
&= \min(\beta_1\beta_2\beta_3\sqrt{t} \cdot \text{UNSAT}(G), \gamma\sqrt{t}\sqrt{t}\frac{1}{t}) \\
&\geq \min(2 \cdot \text{UNSAT}(G), \gamma).
\end{aligned}$$

□

Applying this operation a logarithmic number of times to a constraint graph with a gap of $(1, 1 - \frac{1}{|G|})$ results in a constraint graph with a gap of constant size. The size of this graph is only increased by a polynomial factor, because in each step the size of a graph is only increased by a constant factor. We make this precise in the proof of the PCP theorem.

Theorem 4.6 ([9])**.** *For some $0 < \alpha' < 1$, there exists a reduction from 3SAT to GAP$[1, \alpha']$-CG.*

Proof. It follows from lemma 4.9 that we can transform any instance I of any **NP** problem Π into a constraint graph $G_0 = ((V_0, E_0), \Sigma, \mathcal{C}_0)$, with $|\Sigma| = 3$, such that if I is satisfiable, $\text{UNSAT}(G_0) = 0$ and if I is not satisfiable, $\text{UNSAT}(G_0) > 0$.

We transform this constraint graph into a new constraint graph with a gap of constant size γ from proposition 4.27. We consider repeating our steps $k = log|E_0| = O(log(Size(G)))$ times. For each $1 \leq i \leq k$, let $G_i = step(G_{i-1})$. Clearly, if $\text{UNSAT}(G_0) = 0$, then for each i, $\text{UNSAT}(G_i) = 0$ and specifically $\text{UNSAT}(G_k) = 0$. If $\text{UNSAT}(G_0) > 0$, then $\text{UNSAT}(G_0) \geq \frac{1}{|E_0|}$. By a simple reduction argument, we can proof that $\text{UNSAT}(G_i) \geq \min(2^i\text{UNSAT}(G_0), \gamma)$. We now have

$$\text{UNSAT}(G_k) \geq \min(2^k\text{UNSAT}(G_0), \gamma) = \min(|E_0|\frac{1}{|E_0|}, \gamma) = \gamma$$

The theorem now holds for $\alpha' = 1 - \gamma$. □

For the final result in this section, we follow a standard reduction as shown in [4] to reduce the set of constraint graphs to a set of 3CNF formulas. This set also has a gap of constant size.

Theorem 4.5. *For some $0 < \alpha' < 1$, there exists a reduction from 3SAT to GAP[1,α]-3SAT.*

Proof. Let ϕ be a 3CNF formula. By theorem 4.6, we can reduce this formula into a $\mathsf{GAP}[1, \alpha']$-CG instance $G = ((V, E), \Sigma_0, \mathcal{C})$. We transform this constraint graph into a 3CNF formula with a smaller, but still constant gap size.

For each vertex $v \in V$, we introduce three new binary variables x_{v1}, x_{v2}, x_{v3}, where x_{vi} represents the value of the i-th bit of an assignment for v. Given an edge $e = (v, v') \in E$ the constraint $c(e)$ accepts if and only if $(x_{v1}x_{v2}x_{v3}, x_{v'1}x_{v'2}x_{v'3}) \in c(e)$. We express this condition as a set of $6CNF$ clauses. Let ψ_e be the following 6CNF formula. For each assignment $a \in \{0, 1\}^6$ such that $(a_1a_2a_3, a_4a_5a_6) \notin c(e)$, add a clause of

the six variables, where a variable x is negated in the clause if $a(x) = 1$. By construction, if $a \in \{0,1\}^6$ does not satisfy $c(e)$, then at least one clause of ψ_e is not satisfied. Note that ψ_e contains at most 2^6 clauses.

Now define the 6CNF formula $\phi' := \bigcup_{e \in E} \psi_e$. Given a satisfying assignment $a : V \to \{0,1\}^3$ for G, we define an assignment for the set of variables of ϕ'. For each $v \in V$, let the values of x_{v1}, x_{v2} and x_{v3} be $a(v)_1$, $a(v)_2$, $a(v)_3$ respectively, where $a(v)_i$ is the i-th bit of $a(v)$. For each $e = (v, v') \in E$, we have that $(a(v), a(v')) \in c(e)$, so there is no clause in ψ_e that is not satisfied by a. The formula ϕ' is satisfiable because a is consistent on each v.

If G is at most α-satisfiable, then for any assignment an α fraction of $c(e) \in E$ are not satisfied, so an α fraction of ψ_e's of ϕ' are not satisfied. Since for each edge, we only add 2^6 clauses, this is still a constant fraction of clauses in ϕ.

It is easy to transform this 6CNF formula ϕ' of m clauses into a 3CNF formula ϕ of $4m$ clauses. Consider a clause $l_{v1} \vee l_{v2} \vee l_{v3} \vee l_{v'1} \vee l_{v'2} \vee l_{v'3}$, where l_{vi} is a literal, so either x_{vi} or its negation, for each $v \in V$ and $i \in \{1,2,3\}$. We replace this clause with the clauses $l_{v1} \vee l_{v2} \vee y_1$, $\overline{y_1} \vee l_{v3} \vee y_2$, $\overline{y_2} \vee l_{v'1} \vee y_3$ and $\overline{y_3} \vee l_{v'2} \vee l_{v'3}$. Clearly, an assignment to the variables x_{vi} satisfying ϕ' also satisfies ϕ. Any clause in ϕ' not being satisfied by some assignment a results in four clauses of which one is not satisfied by a. So, the number of unsatisfiable clauses in an unsatisfiable instance is only reduced by a constant factor.

Any constraint graph that is satisfiable is reduced to a 3CNF formula that is satisfiable. A constraint graph that is at most α-satisfiable is reduced to a 3CNF formula that is at most α-satisfiable, for some constant α. □

Lemma 4.30. *Let β be the reduction from theorem 4.6. There exists a polynomial time witness transformation for β that transforms a satisfying assignment for a 3CNF formula ϕ into a satisfying assignment for the constraint graph $\beta(\phi)$.*

Proof. The proof of lemma 4.15 contains a linear-time witness transformation, the witness for the second preprocessing step does not change as seen in the proof of lemma 4.16. In proposition 4.18, the size of the constraints increases by a factor $O(|\Sigma|^{d^t})$. However, since d and t are constant and the size of Σ remains bounded, the size of the constraints and thus the size of the witness is only increased by a constant factor. By lemma 4.22, the reduction $G \circ \mathcal{P}$ also admits a linear time witness reduction. We conclude that we can transform a satisfying assignment for a constraint graph G into a satisfying assignment for $G' := (prep(G))^t \circ \mathcal{P}$ in linear time.

Now, applying this witness transformation k times, for k from proof of theorem 4.6, results in a polynomial-time witness transformation that outputs a satisfying assignment for the resulting graph. □

Lemma 4.31. *Let β be the reduction from GAP$[1, \alpha']$-CG to GAP$[1, \alpha]$-3SAT as described in the proof of theorem 4.5. There exists a polynomial time witness transformation for β that transforms a satisfying assignment for a constraint graph G into a satisfying assignment for the 3CNF formula $\beta(G)$.*

Proof. Let $G = ((V, E), \Sigma, \mathcal{C})$ be a satisfiable constraint graph, let ϕ be the 3CNF formula resulting from the reduction and let $a : V \to \{0,1\}^3$ be a satisfying assignment for G.

We define assignment b as follows. For each $v \in V$, let $b(x_{v1}) := a(v)_1$, $b(x_{v2}) := a(v)_2$ and $b(x_{v3}) := a(v)_3$. Let b take arbitrary values on the rest of the variables. This assignment satisfies ϕ. □

PCP theorem

We have shown that there exists a gap producing reduction from 3SAT to GAP$[1, \alpha]$-3SAT and thus from any language in **NP** to GAP$[1, \alpha]$-3SAT. The existence of this reduction is sufficient the purposes of this thesis this is enough. However, because of the impact of the PCP theorem, we provide a short sketch of the proof as given by Dinur. The reduction can be used to prove the version of the PCP theorem that states that for each language in **NP**, there exists a PCP verifier that uses logarithmic randomness and has constant query complexity.

Proof of theorem 4.4. Let L be a language in **NP** and let β be the gap-producing reduction from L to $\mathsf{GAP}[1,\alpha]\text{-}\mathsf{3SAT}$ as described in corollary 4.5. We construct a PCP verifier for L. Given an instance $x \in \{0,1\}^*$, define the formula $\phi := \beta(x)$. A valid proof for the verifier is an assignment to the variables of ϕ. The verifier chooses one clause uniformly at random and queries the values of this clause.

- (Completeness) If $x \in L$, then ϕ is satisfiable and there exists a satisfying assignment. This proof causes the verifier to accept with a probability of 1.
- (Soundness) If $x \notin L$, then ϕ is at most α-satisfiable. So, given an assignment a to the variables of ϕ, the probability that the verifier picks a clause that is not satisfied by a is at least $1-\alpha$. So, the probability that the verifier rejects is at least $1-\alpha$, which is a constant.

We can repeat this process a constant number of times to achieve a soundness of $\frac{1}{2}$.

The verifier queries only 3 bits, so the repetition queries a constant number of bits. To see that the verifier uses logarithmic randomness, we observe that the number of clauses m in ϕ is polynomial in the size of x and that we require $O(log(m))$ random bits to query an integer in $[1,m]$ uniformly at random. Since $O(log(poly(|x|))) = O(log(|x|))$, we conclude that the verifier uses logarithmic randomness. □

4.3 Optimal inapproximability of MAX-3SAT

We have constructed a set of 3CNF formulas with a gap of $(1,\alpha)$, for some $0 < \alpha < 1$. Håstad [12] shows that this gap can be improved to a gap of $(1-\epsilon, \frac{7}{8}+\epsilon)$, for any sufficiently small ϵ. The reduction from Håstad goes as follows.

First, Håstad reduces the set of 3CNF formulas to a set of more structured 3CNF formulas where each variable appears in exactly 5 clauses. This reduction is constructed such that the new set retains a constant gap size.

Next, Håstad introduces two different probabilistic verifiers that test whether such a 3CNF formula is satisfiable. The first is a verifier with access to two oracles and the second is a PCP verifier with logarithmic randomness that queries only three bits.

By writing down all possible random strings that the PCP verifier uses, Håstad constructs a system of equations over these three bits. Using the properties of the verifier with access to two oracles, Håstad shows that this system has a gap of $(1-\epsilon, \frac{1}{2}+\epsilon)$, for any sufficiently small ϵ. Furthermore, we show that the size of this gap is optimal for this type of system.

Finally, using a standard reduction in PCP theory, Håstad transform the resulting set of system of linear equations with a gap of $(1-\epsilon, \frac{1}{2}+\epsilon)$ into a set of 3CNF formula with a gap of optimal size, namely a gap of $(1-\epsilon, \frac{7}{8}+\epsilon)$, for any sufficiently small ϵ. The optimality of this gap is based on the assumption that $\mathbf{P} \neq \mathbf{NP}$.

4.3.1 Transforming 3-SAT to 3-SAT5

The verifier with access to two oracles that Håstad constructs randomly picks a clause and a variable in that clause. We require that the following methods result in the same probability distribution for any pair of a clause and a variable in that clause.

- Picking a clause uniformly at random and a variable in that clause uniformly at random.
- Picking a variable uniformly at random and a clause containing that variable uniformly at random.

This property is obtained by transforming a 3CNF formula into a formula where every clause contains exactly 3 literals and every variable appears in exactly 5 different clauses. We will call such a formula a *3CNF5* formula and for any $0 \leq s < c \leq 1$ we will write GAP[c,s]-$\mathsf{3SAT5}$ for the gap problem of 3CNF5 formulas with the gap (c,s).

There exists a reduction that transforms 3CNF formulas into 3CNF5 formulas. We simply replace each instance of a variable x with a new variable x_i and enforce equality of the variables representing x with the clauses $x_i \vee \overline{x_{i+1}}$ and $\overline{x_i} \vee x_{i+1}$. This new formula contains each variable 5 times in different clauses and we use dummy variables to increase the size of each clause to 3.

This reduction, however, does not guarantee that the gap size remains constant. For this reason, we first transform a 3CNF formula into a 3CNF formula where every variable appears at most B times, for some constant B. This *3CNFB* formula is then transformed into a 3CNF5 formula. For any $0 \leq s < c \leq 1$, we write GAP[c,s]-3SATB for the gap problem of 3CNFB formulas with gap (c, s).

Due to Feige [10], there exists a gap-preserving reduction from 3CNFB formulas to 3CNF5 formulas. Feige sketches a proof, we work it out in more detail.

Proposition 4.32. *Let $0 < c < 1$ be a constant. For some constant $0 < k < 1$, there exists a reduction from GAP[1,c]-3SATB to GAP[1,kc]-3SAT5.*

Proof. Let ϕ be a 3CNFB formula over some variable set X. We transform ϕ into a 3CNF5 formula ϕ' over some variable set X' in the following manner.

For every variable $x \in X$ we replace each of the b_x occurrences of x in ϕ with a new variable x_i, where $0 \leq i \leq b_x - 1$.

Let ϕ' contain the clauses of ϕ, over the new variable set. Additionally, for each $0 \leq i \leq b_x - 1$, we add the equality clauses $\overline{x_i} \vee x_{i+1}$ and $x_i \vee \overline{x_{i+1}}$, where we compute $i + 1$ modulo b_x.

we have replaced each variable occurring b times with b new variables and for every new variable we have added two clauses ensuring equality between these variables. In the resulting formula, each variable appears exactly 5 times and no variable appears in the same clause more than once.

However, some clauses still contain only 2 literals. We solve this issue by introducing dummy variables variables y, z_1, z_2. We then add $\overline{y}$ to the clause of length 2 and add the extra clauses $y \vee z_1 \vee z_2$, $y \vee z_1 \vee \overline{z_2}$, $y \vee \overline{z_1} \vee z_2$ and $y \vee \overline{z_1} \vee \overline{z_2}$.

Now, ϕ' is a 3CNF formula where no variable appears in the same clause more than once. Some dummy variables (of the type z_1 and z_2) still appear only 4 times. We add dummy variables w until the total number of variables is divisible by 3 and add dummy clauses containing different non-negated dummy literals of the type z_1, z_2 or w, until each dummy variable appears exactly 5 times.

Note that for each clause of ϕ we add a constant number of clauses to ϕ'. So, the number of clauses increases by at most a constant factor. We show that the new instances have the desired gap. Let β denote the reduction that on input ϕ constructs ϕ'.

Lemma 4.33. *There exists a witness transformation for β that transforms a satisfying assignment for a 3CNFB formula into a satisfying assignment for the 3CNF5 formula $\phi' = \beta(\phi)$.*

Proof. Let a be a satisfying assignment for ϕ. First, assign the value $a(x)$ to each x_i that replaces the variable x. Next, assign all dummy variables the value 1. This assignment satisfies ϕ'. □

Lemma 4.34. *If ϕ is at most c-satisfiable, then ϕ' is at most $c \cdot d$-satisfiable, for some constant $d > 1$.*

Proof. The dummy clauses can always be satisfied by setting the dummy variables to 1. Since there are only a constant number of dummy clauses, this brings down the fraction of unsatisfied clauses by only a constant factor.

Consider the clauses containing x_i's or y's. Letting a clause of the form $\overline{x_i} \vee x_{i+1} \vee \overline{y}$ equate to true

by setting y to 0 only reduces the number of unsatisfied clauses by a constant factor, since this results in 1 of the 4 clauses $y \vee z_1 \vee z_2$ is not being satisfied.

Now only the clauses of the type $\overline{x_i} \vee x_{i+1} \vee \overline{y}$, $x_i \vee \overline{x_{i+1}} \vee \overline{y}$ and the original clauses of ϕ remain. Let a be an assignment with the largest number of clauses satisfied, which is at least c. We show that the number of unsatisfiable remaining clauses can only be decreased by a constant factor compared to that of a.

All clauses $\overline{x_i} \vee x_{i+1} \vee \overline{y}$, $x_i \vee \overline{x_{i+1}} \vee \overline{y}$ are satisfied, so our only strategy for improving the number of satisfied clauses is to look at changing some new variables that represent the same original variable to be unequal to the rest. Now, the only method to increase the number of satisfied clauses is to change multiple new variables x_i within a range of indices. We call this process of negating multiple new variables representing the same original variable a *swap*. For example, a swap negates the values of $a(x_i)$ for $i \in [i_1, i_2]$.

A swap results in at least 2 of the 4 equality clauses to be not satisfied. Since the equality clauses are cyclic, at most half of the variables can be swapped to increase the number of satisfied clauses. For the sake of contradiction, assume that swapping more than half of the clauses results in more extra satisfied clauses than swapping at most half. Then there exists an assignment for ϕ, namely a with the value of x negated, that satisfies clauses than a. This is contradiction with the fact that a is an optimal assignment.

Since the number of occurrences of a variable is bounded by B, at most $\frac{B}{2}$ extra clauses are satisfied by a swap. Note that any assignment for ϕ causes at least cm clauses to be unsatisfied. This means we have would have to do at least $\lceil \frac{2cm}{B} \rceil$ swaps to make all of ϕ's clauses in ϕ' be satisfied. Since each swap causes 2 equality clauses to be unsatisfied, we have at least $2 \cdot \lceil \frac{2cm}{B} \rceil \geq (4\lceil \frac{1}{B} \rceil)cm$ unsatisfied clauses in ϕ'. So, the lemma holds for $d = (4\lceil \frac{1}{B} \rceil) > 1$. □

□

The analysis of proposition 4.32 falls apart when the number of occurrences of a variable is not bounded: it is possible to sacrifice some number of equality clauses to satisfy a (potentially) unbounded number of clauses. If we reduced any set of 3CNF formulas to 3CNF5 formulas in this manner, we are not guaranteed that the resulting set has a constant sized gap.

Papadimitriou and Yannakakis [16] show that there exists a reduction from 3SAT to 3SATB that retains a constanta gap size.

Proposition 4.35. *There exists a gap preserving reduction from GAP[1,c]-3SAT to GAP[1,c']-3SATB, where c is the constant from corollary 4.5 and c′ is a constant factor of c.*

In their proof, Papadimitriou and Yannakakis use expander graphs to ensure that negating some number of variables requires us to sacrifice a high number of equality clauses. For the completeness of this section, we repeat the relevant definition for expander graphs from section 4.2.

Given a graph $G = \{V, E\}$ and two subsets of vertices $S, T \subset V$ we denote $E(S,T) = |S \times T \cap E|$ for be the number of edges between S and T. If S and T partition V, we call the set $S \times T \cap E$ a *cut* of the graph. For a subset of nodes $S \subset V$ we denote $\overline{S} := V \setminus S$ for the complement of S.

Definition 4.18. The expansion rate of a d-regular graph $G = \{V, E\}$ is defined as:

$$h(G) = \min_{S:|S| \leq |V|/2} \frac{E(S, \overline{S})}{|S|}$$

The reduction of proposition 4.35 relies on the connectivity of expander graphs. Similar to the reduction from 3CNFB to 3CNF formulas, we introduce new variables x_i. Equality is ensured in a different manner. We construct a highly connected graph with nodes x_i and add the equality clauses $x_i \vee \overline{x_j}$ and $\overline{x_i} \vee x_j$ when the variables x_i and x_j are connected. We construct the graph such that changing the value of some number of new variables, compared to the best assignment, results in a high number of equality clauses that are not

satisfied. We will work out the details in the proof of proposition 4.35.

We construct an expander graph to achieve this high connectivity.

Lemma 4.36 ([3])**.** *Given $n \in \mathbb{N}$, there exists some $c \in \mathbb{R}_{>0}$, such that in polynomial we can construct a 3-regular graph $G = \{V, E\}$, with expansion rate $h(G) \geq c$ and $|V| = n$.*

Using this expander graph, Papadimitriou and Yannakakis construct the following graph. For a variable with m occurrences, the graph will contain $O(m)$ nodes to ensure the connectivity of the nodes.

Lemma 4.37. *Given $m \in \mathbb{N}$, in polynomial time we can construct a graph $F_m = \{V, E\}$ with the following properties:*

1. *F_m has bounded degree.*
2. *F_m has $O(m)$ total nodes, including a set D of m "distinguished" nodes.*
3. *Let $(T, \overline{T})$ be a partition of V and let $S_1 = D \cap T$ and $S_2 = D \cap \overline{T}$ be the sets of distinguished nodes on each side of the partition. It holds that $E(T, \overline{T}) \geq \min(|S_1|, |S_2|)$.*

Proof. Fix some $0 < c < 1$. We construct the graph F_m as follows: Take m disjoint full binary trees with at least $\frac{1}{c}$ leaves and connect the leaves with the edges of a 3-regular graph with expansion rate c. The resulting graph has (1) bounded degree and (2) $O(m)$ nodes, where the set D of distinguished nodes is the set of roots of the trees. We now show that F_m satisfies property (3).

Let $(T, \overline{T})$ be a partition of V and let $S_1 = D \cap T$ and $S_2 = D \cap \overline{T}$ be the sets of distinguished nodes on each side of the partition. Let R_1, R_2 be the set of distinguished nodes whose trees are entirely contained in respectively $T, \overline{T}$ and let R_3 be the rest of the distinguished nodes.

Let V' be the set of leaves and let $G' = \{V', E'\}$ be the expander graph on the leaves. Note that within G' the $(V' \cap T, V' \cap \overline{T})$ is a partition. By definition 4.18 it holds that

$$c \leq h(G') \leq \frac{E(V' \cap T, V' \cap \overline{T})}{\min(|V' \cap T|, |V' \cap \overline{T}|)} \leq \frac{E(V' \cap T, V' \cap \overline{T})}{\min(\frac{1}{c}|R_1|, \frac{1}{c}|R_2|)}$$

Where the last inequality comes from the fact that $V' \cap T$ contains at least the $\frac{1}{c}$ leaves from the $|R_1|$ number of trees entirely contained in T. In the same way it holds that $|V' \cap \overline{T}| \geq \frac{1}{c}|R_2|$. Now we have the following inequality about the number of edges from the expander in the cut:

$$E(V' \cap T, V' \cap \overline{T}) \geq \min(|R_1|, |R_2|) \tag{6}$$

This is a lower bound for the number of edges from the expander graph that are in the cut.

We now look at a lower bound for the number of edges from the trees that are in the cut. For every distinguished node whose tree is not entirely contained in T or $\overline{T}$ at least one edge from the tree is contained in the cut. This means that there are at least $|R_3|$ edges from trees in the cut. So

$$E(V' \cap T, V'^c \cap \overline{T}) + E(V'^c \cap T, V' \cap \overline{T}) + E(V'^c \cap T, V'^c \cap \overline{T}) \geq |R_3|$$

We need the following technical lemma

Lemma 4.38. *Given a graph $G = \{V, E\}$ and a partition $T, \overline{T}$ and a set of vertices $V' \subset V$, we have that*

$$E(T, \overline{T}) = E(V' \cap T, V' \cap \overline{T}) + E(V' \cap T, V'^c \cap \overline{T}) + E(V'^c \cap T, V' \cap \overline{T}) + E(V'^c \cap T, V'^c \cap \overline{T})$$

Proof. This lemma follows from the definition of $E(T, \overline{T})$ and the fact that $(V' \cap T)$, $(V'^c \cap T)$, $(V' \cap \overline{T})$ and $(V'^c \cap \overline{T})$ partition V. □

Using equations (6) and (4.3.1), and lemma 4.38, we now have that

$$\begin{aligned} E(T,\overline{T}) &= E(V'\cap T, V'\cap \overline{T}) + E(V'\cap T, V'^c\cap \overline{T}) + E(V'^c\cap T, V'\cap \overline{T}) + E(V'^c\cap T, V'^c\cap \overline{T}) \\ &\geq \min(|R_1|,|R_2|) + |R_3| \\ &\geq \min(|S_1|,|S_2|) \end{aligned}$$

□

We are now in the position to prove proposition 4.35.

Proof of proposition 4.35. For each variable x that occurs m times we add the $O(m)$ variables $x_1, x_2, \ldots$ of F_m. We replace the occurrences of x in ϕ with the distinguished variables and for each edge (x_i, x_j) in F_m, we add the equations $\overline{x_i} \vee x_j$ and $x_i \vee \overline{x_j}$. Since F_m has bounded degree and each edge adds two clauses, the number of clauses of ϕ' increases by at most a constant factor compared to ϕ. Let β be the gap-preserving reduction that on input ϕ constructs ϕ'.

Lemma 4.39. *There exists a witness transformation for β that transforms a satisfying assignment for a 3CNF formula into a satisfying assignment for the 3CNFB formula $\phi' = \beta(\phi)$.*

Proof. Let a be a satisfying assignment for ϕ. Assign the value $a(x)$ to each x_i that replaces the variable x. This assignment satisfies ϕ'. □

Lemma 4.40. *If ϕ is at most c-satisfiable, then ϕ' is at most $c \cdot d$-satisfiable, for some constant $d > 1$.*

Proof. Let a be a an assignment with the largest number of clauses satisfied. Let a' be the corresponding assignment for ϕ' where every new variable x_i is assigned the value $a(x)$.

All equality clauses are satisfied. So, the only method to increase the number of clauses satisfied by a' is to swap some new variables x_i to the negation of x. We observe that negating a set of variables x_i corresponds to a partition $(T, \overline{T})$ of the variables. Let S_1 and S_2 be the variables on each side of the partition corresponding to the distinguished nodes, these are precisely the variables that are contained in the the clauses of ϕ' corresponding to the clauses of ϕ. Assume without loss of generality that $|S_1| \leq |S_2|$. The variables of S_1 are negated, since if the variables of S_2 can be negated resulting in less clauses unsatisfied, there would be an assignment for ϕ with less variables unsatisfied, namely a, but with x negated.

$E(T, \overline{T})$ is equal to the number of edges in the cut, representing the equality clauses now unsatisfied after a swap. By property (3) of the graph F_m, we have that $E(T, \overline{T}) \geq |S_1|$. This means that if we want to satisfy $|S_1|$ extra clauses by negating S_1, we sacrifice at least $|S_1|$ equality clauses. So, no swap of variables can result in an increase of satisfied clauses. The number of unsatisfied clauses of ϕ is a constant factor of the number of unsatisfied clauses of ϕ, since the length of ϕ' is only a constant factor longer than the length of ϕ. □

□

4.3.2 Two prover one round systems for 3-SAT5

In a two prover one round system, we run a polynomial-time probabilistic Turing machine V, called the verifier, that has access to two oracles, $P_1 : \{1,-1\}^* \to \{1,-1\}^*$ and $P_2 : \{1,-1\}^* \to \{1,-1\}^*$, called the provers. The verifier is allowed to only access the oracles once.

On input $x \in \{1,-1\}^*$, V computes a pair of strings (q_1, q_2), which we interpret as a pair of questions. In constant time complexity V receives the values $P_1(q_1)$ and $P_2(q_2)$, which we interpret as the answers from P_1 and P_2 to the questions. A prover does not have access to the question of the other prover, it only has access to the input of V and its own question. We do not limit the size of the output of the provers. However, since V runs in polynomial time, it will read at most a polynomial number of bits of the output.

We write V^P for the verifier with access to $P = (P_1, P_2)$ and we write $\mathbb{P}[V^P(x) = b]$ for the probability, taken over the random strings r, that V^P outputs $b \in \{1,-1\}$.

Definition 4.19. Let L be a language and let $0 < s \leq c < 1$ be constants. Let V be a verifier in a two prover one round system. We say that V recognizes L with soundness s and completeness c if for any input x, the following conditions hold:

- (Completeness) If $x \in L$, there exists a pair P of provers such that $\mathbb{P}[V^P(x) = 1] \geq c$.
- (Soundness) If $x \notin L$, for every pair P of provers we have that $\mathbb{P}[V^P(x) = 1] \leq s$.

Håstad makes use of a known two prover one round system for 3SAT5. First, we describe a basic two prover one round system that will form the basis of the two prover one round system from Håstad. The verifier takes a 3CNF5 formula ϕ as input and the provers give a proof to show that ϕ is satisfiable.

The input for the verifier V is a 3CNF5 formula $\phi = \{C_i\}_{i=0}^m$ over variable set X, where C_i contains literals of the variables x_{a_i}, x_{b_i} and x_{c_ij}. V is defined by the following algorithm.

Two prover one round verifier V

Input: A boolean formula ϕ over variable set X.
Proof: A pair of provers (P_1, P_2) outputting values for variables in X.

1. Choose $j \in [m]$ and $k \in \{a_j, b_j, c_j\}$ both uniformly at random.
2. Construct the question $(x_{a_j}, x_{b_j}, x_{c_j})$ for P_1 and construct the question x_k for P_2.
3. Accept if $P_1(x_{a_j}), P_1(x_{b_j}), P_1(x_{c_j})$ satisfies C_j and $P_1(x_k) = P_2(x_k)$.

If ϕ is satisfiable, then there exist provers P_1 and P_2 such that the acceptance probability of the verifier is 1. To see this, let $a \in \{0,1\}^n$ be a satisfying assignment for ϕ. Given the question $(x_{a_j}, x_{b_j}, x_{c_j})$, P_1 outputs the values $(a(x_{a_j}), a(x_{b_j}), a(x_{c_j}))$ and given the questions x_k, P_2 outputs the value $a(x_k)$.

If ϕ is not satisfiable, then for any pair of provers $P = (P_1, P_2)$ the acceptance probability of V^P is strictly less than 1.

Lemma 4.41. *Let ϕ be a 3CNF5 formula that is at most c-satisfiable. For any strategy P, V^P accepts with probability at most $\frac{2+c}{3}$.*

Proof. Since the answers for the questions to P_1 and P_2 are returned independently, the value $P_2(x_k)$ is fixed for any $k \in [n]$. For this reason, P_2 fixes the assignment $a : x \to \{0,1\}$ defined by $x \mapsto P_2(x_k)$.

Now let P_1 be any prover. If the verifier chooses a clause not satisfied by a, P_1 has two options.

- Return the values $a(x_{a_j}), a(x_{b_j}), a(x_{c_j})$.
- Change the value of at least one of the three variables such that this new assignment satisfies C_j.

In the first case, the verifier always rejects. In the second case, the probability of the verifier rejecting is at least $\frac{1}{3}$, namely the probability that V chooses a variable whose value is changed by P_2.

The probability of V choosing a clause that is not satisfied by a is at least a $1 - c$. So, the probability of V rejecting is at least $\frac{1-c}{3}$ and the probability of V accepting is at most $\frac{2+c}{3}$. □

This basic two prover one round system for 3SAT5 has completeness 1 and soundness $\frac{2+c}{3}$ on a set of 3CNF5 formulas with a gap of $(1, c)$, for any constant $0 < c < 1$. It is possible to decrease the soundness by repeating the system u times and only accepting when all of the individual runs accept. This results in a soundness of $\frac{2+c}{3}^u$.

However, in this system we ask multiple questions to the provers. To solve this issue, we ask u question to the provers in parallel and only accept when each independent pair of answers accept. The following theorem by Raz [18] shows that if the answer size remains bounded, then the soundness in such a parallel repeated system decreases exponentially with u.

Proposition 4.42 ([18]). *For all $d \in \mathbb{N}$ and $s < 1$, there exists a constant $c_{d,s}$ such that given a two prover one round system with soundness s, completeness 1 and answer size bounded by d, for any $u \in \mathbb{N}$, the soundness of u protocols run in parallel is at most $c_{d,s}^u$.*

In our two prover one round system, the answer size is potentially unbounded. However, we can treat the answer size as bounded, because the verifier reads only a bounded number of bits from the answer. We show a variant of the basic two prover one round system that asks $u \in \mathbb{N}$ questions. This verifier V_u is given as input a 3CNF5 formula $\phi = \{C_i\}_{i=0}^m$ over variable set X. It then computes the following algorithm.

u-parallel two prover one round verifier V_u

Input: A boolean formula ϕ over variable set X.
Proof: A pair of provers (P_1, P_2) outputting values for variables in X.

1. For each $i \in [u]$ choose both $j_i \in [m]$ and $k_i \in \{a_{j_i}, b_{j_i}, c_{j_i}\}$ uniformly at random.
2. Construct the questions $x_{a_{j_i}}$, $x_{b_{j_i}}$ and $x_{c_{j_i}}$ for P_1 and construct the questions x_{k_i} for P_2.
3. Accept if for all $i \in [u]$, $P_1(x_{a_{j_i}}), P_1(x_{b_{j_i}}), P_1(x_{c_{j_i}})$ satisfies C_{j_i} and $P_1(x_{k_i}) = P_2(x_{k_i})$.

We have the following result on the completeness and soundness of this system.

Lemma 4.43. *V_u has completeness 1 and there exists a constant c_c such that the soundness of V_u is c_c^u.*

Proof. For the completeness, similar to the completeness of the basic two prover one round system, let P_1 and P_2 return the values of some assignment a that satisfies ϕ. Now, the verifier accepts with probability 1. The soundness is an immediate result of proposition 4.42 and lemma 4.41. □

4.3.3 PCP system for 3SAT

We set aside the two prover one round system to introduce some theory on long codes and to introduce the PCP verifier. At the end of this section, we show how Håstad makes use of the two prover one round system to provide an upper bound of the soundness of this PCP system.

Definitions and notation
The PCP verifier of Håstad queries three bits and accepts based on the XOR of these three bits. We will write this condition in terms of a linear equation in three variables. For this reason, we work with the binary set $\{1, -1\}$ instead of $\{0, 1\}$, where -1 represents *true* and 1 represents *false*. Binary operations on $\{1, -1\}$ are calculated by their truth value. For example $1 \vee 1 = 1$ and $1 \vee -1 = -1$. Now, the XOR of two binary variables over $\{0, 1\}$ is equivalent with the multiplication of two binary variables over $\{1, -1\}$.

Let $A = \{a_1, \dots a_n\}$ be a finite ordered set and let $f : A \to \{1, -1\}$ be a function. In some cases, it is more practical to work with the truth table of f. For this reason, we identify f with the string $f(a_1) \dots f(a_n)$ of length n.

Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of binary-valued variables. We denote $\{1, -1\}^X$ for the set of all possible assignments of X, that is, all binary strings of size $|X|$, or equivalently, all functions $a : X \to \{1, -1\}$. Let $\mathcal{F}_X := \{f : \{1, -1\}^X \to \{1, -1\}\}$ be the set of functions from $\{1, -1\}^X$ to $\{1, -1\}$.

The PCP verifier of Håstad checks whether a 3CNF formula is satisfied by some assignment. A valid proof will be an encoding of this assignment. Håstad builds this encoding upon the *long code* encoding of an assignment.

Definition 4.20. The *long code* of assignment $a \in \{1, -1\}^X$ is the function $A_a : \mathcal{F}_X \to \{1, -1\}$ defined by $f \mapsto f(a)$.

The long code is an overly redundant encoding. Because of this, we can test the consistency of a string $A : \mathcal{F}_X \to \{1, -1\}$ using a low number of bits. By consistency we mean that each coordinate of A indeed

contains the function value of the same assignment a.

Håstad constructs a verifier that tests whether a function $A : \mathcal{F}_X \to \{1, -1\}$ is a long code of some assignment $a \in X$ and proves that this verifier has large enough soundness by analyzing the Fourier expansion of A. Håstad shows that if A has a small set of large coefficients, then it is a long code of some assignment a. The details of this Fourier analysis is beyond the scope of this thesis and we refer the reader to the analysis of Håstad [12].

Given a set of variables X and some assignment $a \in \{1, -1\}^X$, the long code A_a satisfies $A_a(f) = -A_a(-f)$ for all $f \in \mathcal{F}_X$. We want to enforce this property on an arbitrary function $A : \mathcal{F}_X \to \{1, -1\}$. This helps ensure that there are no large coefficients in the Fourier expansion of A that do not correspond to an assignment.

Håstad achieves this by *folding A over true*. The definition of folding relies on an arbitrary, but fixed method that given a pair of functions $(f, -f)$ picks f or $-f$.

Definition 4.21. Let $A : \mathcal{F}_X \to \{1, -1\}$ be a function. We define the function A_{-1} as follows. For each pair $(f, -f)$, where $f \in \mathcal{F}_X$ and $-f \in \mathcal{F}_X$, pick f or $-f$ and define

- $A_{-1}(f) := A(f)$ and $A_{-1}(-f) := -A(f)$ if we pick f.
- $A_{-1}(f) := -A(-f)$ and $A_{-1}(-f) := A(-f)$ if we pick $-f$.

The function A_{-1} is called *A folded over true*

Given a set of variables X and a function $A : \mathcal{F}_X \to \{1, -1\}$ the following two lemmas clearly hold.

Lemma 4.44. *For any $f \in \mathcal{F}_X$, $A_{-1}(f) = -A_{-1}(-f)$.*

Lemma 4.45. *Let $a \in \{1, -1\}^X$ be an assignment and let A_a be the long code of a. We have that $A_a = A_{-1}$.*

Proof. For any $f \in \mathcal{F}_X$ it holds that either

$$A_{-1}(f) = A(f)$$

or

$$A_{-1}(f) = -A(-f) = - - f(a) = f(a) = A(f)$$

□

The long code of some assignment is unchanged by folding it over true. The PCP verifier that Håstad constructs test whether some function $A : \mathcal{F}_X \to \{1, -1\}$ is the long code of an assignment $a : X \to \{1, -1\}$. Besides this, we want to simultaneously test whether this assignment satisfies some formula ϕ, that we interpret as a function h. We achieve this by *conditioning A over h*.

Definition 4.22. Let $A : \mathcal{F}_X \to \{1, -1\}$ and $h \in \mathcal{F}_X$ be two functions. We define the function A_h as follows. For each $f \in \mathcal{F}_X$, define $A_h(f) = A(f \wedge h)$. Here we define the conjunction of a function by the conjunction of each input: $(f \wedge g)(a) = f(a) \wedge g(a)$ for any functions $f, g \in \mathcal{F}_X$ and assignment $a \in \{0, 1\}^X$.

Håstad show that a long code A_a of some assignment $a : X \to \{1, -1\}$ conditioned over h depends on A_a only if a satisfies h.

Lemma 4.46. *Let $a \in \{1, -1\}^X$ be an assignment and let $A = A_a : \mathcal{F}_X \to \{1, -1\}$ be the long code of a. For all $f, h \in \mathcal{F}_X$, we have that*

$$A_h(f) = \begin{cases} A(f) & \text{if} \quad h(a) = -1 \\ 1 & \text{if} \quad h(a) = 1 \end{cases}$$

Proof. By definition:

$$\begin{aligned} A_h(f) &= A(f \wedge h) && (7)\\ &= (f \wedge h)(a) && (8)\\ &= f(a) \wedge h(a) && (9) \end{aligned}$$

If $h(a) = -1$, we have

$$\begin{aligned} (7) &= f(a) \wedge h(a)\\ &= f(a) \wedge -1\\ &= f(a)\\ &= A(f) \end{aligned}$$

If $h(a) = 1$, then

$$\begin{aligned} (7) &= f(a) \wedge h(a)\\ &= f(a) \wedge 1\\ &= 1 \end{aligned}$$

□

Finally, Håstad defines a method to apply both folding and conditioning over some function h. We make a pairing of every function of the form $f \wedge h$ and every function of the form $(-f) \wedge h$ and fix a method of choosing either one. We write $-f \wedge h$ for $(-f) \wedge h$ when it causes no ambiguity.

Definition 4.23. Let $A : \mathcal{F}_X \to \{1, -1\}$ and $h \in \mathcal{F}_X$ be functions. We define $A_{-1,h}$ as follows. For every pair of functions $(f \wedge h, -f \wedge h)$, pick either $f \wedge h$ or $-f \wedge h$ and define

- $A_{-1,h}(f) := A(f \wedge h)$ and $A_{-1,h}(-f) := -A(f \wedge h)$ if we pick $f \wedge h$.
- $A_{-1,h}(f) := -A(-f \wedge h)$ and $A_{-1,h}(-f) := A(-f \wedge h)$ if we pick $-f \wedge h$.

Lemma 4.47. *Let $a \in \{1, -1\}^X$ be an assignment and let $A = A_a : \mathcal{F}_X \to \{1, -1\}$ be a long code of a. For all functions $f, h \in \mathcal{F}_X$, if $h(a) = -1$, then $A_{-1,h}(f) = A(f)$ and if $h(a) = 1$, then $A_h(f)$ is a constant that is only dependent on the method of folding.*

Proof. If $h(a) = -1$

$$\begin{aligned} A(f \wedge h) &= (f \wedge h)(a)\\ &= f(a) \wedge h(a)\\ &= f(a) \wedge -1\\ &= f(a)\\ &= A(f) \end{aligned}$$

$$\begin{aligned} -A(-f \wedge h) &= -(-f \wedge h)(x)\\ &= -(-f(a) \wedge h(a))\\ &= -(-f(a) \wedge -1)\\ &= -(-f(a))\\ &= f(a)\\ &= A(f) \end{aligned}$$

So, $A_{-1,h}(f) = A(f)$.

If $h(a) = 1$, then

$$\begin{aligned} A(f \wedge h) &= (f \wedge h)(a) \\ &= f(a) \wedge h(a) \\ &= f(a) \wedge 1 \\ &= 1 \end{aligned}$$

and

$$\begin{aligned} -A(-f \wedge h) &= -(-f \wedge h)(a) \\ &= -(-f(a) \wedge h(a)) \\ &= -(-f(a) \wedge 1) \\ &= -1 \end{aligned}$$

So, $A_{-1,h}(f)$ is a constant depending on whether we choose $f \wedge h$ or $-f \wedge h$ in our folding method. □

Håstads PCP verifier
We are now in the position to construct Håstads PCP verifier for 3SAT.

An oracle π in a PCP system with verifier V is often interpreted as a finite, but possibly unbounded string that proves a statement: for each question $q \in \{0,1\}^*$ that can be asked by V, the value $P(q)$ appears in the string. The verifier can query a value in this written proof at any location in constant time.

Before introducing the PCP verifier, Håstad introduces a format for a proof in this system.

Definition 4.24. Let ϕ be a formula with n variables. A *standard written proof* for ϕ with parameter u (SWP_u) is the concatenation of for each set $V \subset [n]$ with $|V| \leq 3u$ a binary string of size $2^{2^{|V|}}$.

For each set $V \subset [n]$, the string of size $2^{2^{|V|}}$ represents some function $A_V : \mathcal{F}_V \to \{0,1\}$. A verifier tests whether there exists a satisfying assignment $a \in \{0,1\}^n$ for ϕ such that for each $V \subset [n]$ with $|V| \leq 3u$, A_V is the long code of $x|_V$.

Definition 4.25. Let ϕ be a formula with n variables. We say that a standard written proof SWP_u for ϕ is a *correct* standard written proof if there exists some assignment $a \in \{0,1\}^n$ that satisfies ϕ, such that for each $V \subset [n]$ of size at most $3u$, A_V is the long code of $x|_V$.

Let $a \in \{0,1\}^n$ be an assignment of some ϕ of n variables. Given u, we can construct a natural SWP_u for ϕ. For each $V \subset [n]$ with $|V| \leq 3u$, let A_V be the long code of $x|_V$. We interpret this function as a string of size $2^{2^{|V|}}$ and let SWP_u be the concatenation of these strings.

Lemma 4.48. *Given u, if $x \in \{0,1\}^n$ satisfies ϕ, then SWP_u as constructed above is a correct SWP for ϕ.*

Proof. SWP_u is the concatenation of A_V's that are the long codes of x, which satisfies ϕ. □

The verifier in our PCP system for 3SAT5 has oracle access to a proof SWP_u and reads only three bits. The verifier, with parameters ϵ and u, takes as input some 3CNF5 formula ϕ. It attempts to decide whether ϕ is satisfiable, by determining whether the given proof is a correct standard written proof for ϕ.

Using a low number of bitsThe formula ϕ has the form $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, with C_j containing the variables $x_{a_j}, x_{b_j}, x_{c_j}$. We define the verifier $L^\epsilon(u)$ by the following algorithm.

Verifier $L^\epsilon(u)$

Input: A boolean formula ϕ.
Proof: A SWP with parameter u.

1. For $i \in [u]$, pick $j_i \in [m]$ and $k_i \in \{a_{j_i}, b_{j_i}, c_{j_i}\}$ uniformly at random.
2. Set $U := \{k_1, k_2, \ldots, k_u\}$, set $W := \bigcup_{i=1}^{u} \{a_{j_i}, b_{j_i}, c_{j_i}\}$ and define $h := \bigwedge_{i=1}^{u} C_{j_i}$.
3. Pick both $f \in \mathcal{F}_U$ and $g_1 \in \mathcal{F}_W$ uniformly at random.
4. Define the function $\mu \in \mathcal{F}_W$ as follows: for each $y \in \{0,1\}^W$, let $\mu(y) = 1$ with probability $1 - \epsilon$ and let $\mu(y) = -1$ with probability ϵ.
5. Define the function $g_2 \in \mathcal{F}_W$ as follows: let $g_2 = f g_1 \mu$, that is, for every $y \in \{0,1\}^W$, let $g_2(y) = f(y|_U) g_1(y) \mu(y)$.
6. Query from the oracle $\pi := SWP_u$ the values of the following bits: $(A_U)_{-1}(f)$, $(A_W)_{-1,h}(g_1)$ and $(A_W)_{-1,h}(g_2)$.
7. Accept if $\pi((A_U)_{-1}(f))\ \pi((A_W)_{-1,h}(g_1))\ \pi((A_W)_{-1,h}(g_2)) = 1$.

Lemma 4.49. *The completeness of $L^{\epsilon}(u)$ is $1 - \epsilon$.*

Proof. Let ϕ be a satisfiable 3CNF5 formula with n variables and let $a \in \{0,1\}^n$ be a satisfying assignment for ϕ. By lemma 4.48 we can construct a correct SWP_u for ϕ. Now by lemmas 4.45 and 4.47 and the definition of g_2, we have that

$$(A_U)_{-1}(f) \cdot (A_W)_{-1,h}(g_1) \cdot (A_W)_{-1,h}(g_2) = f(x|_W) g_1(x|_W) f(x|_W) g_1(x|_W) \mu(x|_W) = \mu(x|_W)$$

The value $\mu(x|_W)$ is equal to 1 with probability $1 - \epsilon$. So, if ϕ is satisfiable, the verifier can be made to accept with at least probability $1 - \epsilon$. □

Håstad shows that if the soundness of the system is high enough, then we can make the u-parallel two prover one round system system accept with high probability.

Lemma 4.50 ([12])**.** *For any sufficiently small $\epsilon, \delta > 0$, if $\mathbb{P}_r[L^{\epsilon}(u)$ accepts$] \geq \frac{1+\delta}{2}$ for some SWP_u, then there is a strategy for P_1 and P_2 that makes the u-parallel two prover one round system system accept with probability at least $4\epsilon\delta^2$.*

Systems of binary linear equations
We use this PCP system to reduce the set of 3CNF5 formulas to the problem problem of solving a system of equations with binary valued variables. The important contribution of Håstad in [12] is constructing this reduction and showing that it increases the gap size to an optimal value.

The set of systems with an optimal gap size is then reduced to a set of 3CNF formulas with an optimal gap size. We introduce the problem of satisfying a specific type of system of linear equations, as well as a maximization version and a gap version.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of binary variables. We will only work with systems of binary linear equations with exactly 3 variables. The system then contains linear equations of the form $x_i x_j x_k = b$, with $b \in \{1, -1\}$ being a constant. Note that this system is allowed to be inconsistent. An assignment $a : X \to \{1, -1\}$ satisfies a system of binary linear equations L if the assignment satisfies all equations in the system.

The language 3LIN2 contains (encodings of) satisfiable systems of binary linear equations over exactly 3 variables. The maximization version MAX-3LIN2 is the problem of maximizing the number of equations simultaneously satisfied by an assignment.

Let L be a system of m binary linear equations. We say that L is c-satisfiable if $\mathrm{OPT}(L) \geq cm$ and

we say that L is s-satisfiable if $\mathrm{OPT}(L) \leq sm$. For any $0 \leq s < c \leq 1$, the gap problem GAP[c,s]-3LIN2 is defined by the pair of at least c-satisfiable and at most s-satisfiable systems of binary linear equations over exactly 3 variables.

Inapproximability of MAX-3LIN2
The most important contribution from Håstad is reducing a set of 3CNF formulas with a small gap to a set of system of binary linear equations over exactly 3 variables with an optimal gap. From this result we can conclude the optimal inapproximabilty of this problem, as well as that of multiple other problems. Most notably MAX-3SAT.

For each internal random string r of the verifier $L_\epsilon(u)$, we have an acceptance condition of in the form of an equation, namely the XOR of three bits. Håstad lists each in a system of linear equations and shows that this process results in a set of systems with the desired gap.

Proposition 4.51. *For sufficiently small $\delta > 0$ there exists a reduction from GAP[1,k]-3SAT5 to GAP[$1-\delta, \frac{1+\delta}{2}$]-3LIN2 where k is the constant from proposition 4.32.*

Proof. Let ϕ be a 3CNF5 formula. Set u such that $c^u < 4\delta^3$, where c is the constant from proposition 4.42.

Let $n := |\phi|$ and let SWP_u be a proof for ϕ. For each bit b in SWP_u we introduce the variable x_b. We consider applying the test $L^\delta(u)$ to ϕ. We denote the bits corresponding to $(A_U)_{-1}(f)$, $(A_W)_{-1,h}(g_1)$ and $(A_W)_{-1,h}(g_2)$ by $x_{b_{U,f}}$, $x_{b_{W,h,g_1}}$ and $x_{b_{W,h,g_2}}$ respectively. The acceptance condition for the test is equivalent to $x_{b_{U,f}} x_{b_{W,h,g_1}} x_{b_{W,h,g_2}} = b'$, where b' is a constant. This constant is dependent on our method of folding.

We create a system of linear equations L as follows. For every possible internal random string r the verifier uses, we add the linear equation $x_{b_{U,f}} x_{b_{W,h,g_1}} x_{b_{W,h,g_2}} = b'$ to our system. A SWP now defines an assignment to the variables of the system.

We now show that this system is either at least $(1-\delta)$-satisfiable or at most $\frac{1+\delta}{2}$-satisfiable.

Let ϕ be a satisfiable 3CNF5 formula and let $a \in \{0,1\}^n$ be a satisfying assignment for ϕ. Let SWP_u be a correct standard written proof as given by lemma 4.48. By lemma 4.49, L^δ accepts with probability $1-\delta$ on SWP_u. Let the values of the variables $x_{b_{U,f}}$, $x_{b_{W,h,g_1}}$ and $x_{b_{W,h,g_2}}$ be determined by SWP_u, based on its values for A_U and A_W. Each equation in L represents a run of $L^\delta(u)$ on a different random string. Since the probability that $L^\delta(u)$ accepts on SWP_u is at least $1-\delta$, at least a $1-\delta$ fraction of equations in L are satisfied by the assignment given by SWP_u.

Let ϕ be a 3CNF5 formula that is at most $(1-k)$-satisfiable for some constant $0 < k < 1$. By proposition 4.42, the probability that the u-parallel two prover one round system accepts on input ϕ is at most c^u, for some constant $0 < c < 1$. Assume for the sake of contradiction that L is at least $(\frac{1+\delta}{2})$-satisfiable. By lemma 4.50 there exists a strategy for P_1 and P_2 such that the u-parallel two prover one round system accepts with probability at least $4\delta^3$. This is a contradiction with the fact that $c^u < 4\delta^3$. So, the system is at most $(\frac{1+\delta}{2})$-satisfiable.

We conclude that the set of systems of binary linear equations over exactly 3 variables we have constructed has a gap of $(1-\delta, \frac{1+\delta}{2})$. □

Corollary 4.51.1. *For any $\epsilon > 0$, there exists no polynomial time $(\frac{1}{2}+\epsilon)$-approximation algorithm for MAX-3LIN2, unless $\boldsymbol{P = NP}$.*

Proof. Set δ to a negative power of 2 such that $\frac{(1+\delta)/2}{1-\delta} < \frac{1}{2} + \epsilon$. Let L be any language, let $x \in \{0,1\}^*$ be an instance. Let $\phi = \phi_x$ be a 3CNF5 formula resulting from the Cook-Levin reduction, the PCP theorem, proposition 4.35 and proposition 4.32 and let $M := M_\phi$ be the system of linear of linear equations resulting from theorem 4.51.

Assume there exists a $(\frac{1}{2}+\epsilon)$-approximation algorithm for MAX-3LIN2. Consider the following cases

- If $x \in L$, then M is at least $(1-\delta)$-satisfiable. Now A outputs an assignment that satisfies at least $(1-\delta)(1-\epsilon)$ clauses.
- If $x \notin L$, then M is at most $\frac{1+\delta}{2}$-satisfiable.

Since $\frac{(1+\delta)/2}{1-\delta} < \frac{1}{2}+\epsilon$, we have that $(1+\delta)/2 < (1-\delta)(\frac{1}{2}+\epsilon)$. So, such an algorithm can be used to decide whether $x \in L$ in polynomial time. This is not possible unless $\mathbf{P} = \mathbf{NP}$. □

In the proof of theorem 4.51, Håstad constructs a witness transformation for the reduction.

Lemma 4.52. *Let β be the reduction from theorem 4.51. There exists a polynomial time witness transformation for β that transforms a satisfying assignment for a 3CNF5 formula ϕ into a satisfying assignment for the system of linear equations $\beta(\phi)$.*

Proof. Let ϕ be a satisfiable 3CNF5 formula and let L_ϕ be the resulting system of linear equations from proposition 4.51. We have seen that a satisfying assignment for ϕ can be transformed into an accepting SWP for ϕ. This SWP can again be transformed into an assignment for L_ϕ. □

4.3.4 MAX-3LIN2 to MAX-3SAT

The final reduction that Håstad introduces is standard in PCP theory.

Theorem 4.53. *For any sufficiently small $\delta > 0$, there exists a reduction from GAP$[1-\delta, \frac{1+\delta}{2}]$-3LIN2 to GAP$[1-\frac{3}{4}\delta, \frac{7}{8}+\frac{1}{8}\delta]$-3SAT.*

Proof. Let L be a system of linear equations over variable set X with $|L| := m$. We create a 3CNF formula ϕ over this same variable set in the following manner. For every equation $x_i x_j x_k = b$ we add 4 clauses to ϕ. We separate two cases.

- If b is 1, then we add the following clauses:
  $\overline{x_i} \vee x_j \vee x_k, \quad x_i \vee \overline{x_j} \vee x_k, \quad x_i \vee x_j \vee \overline{x_k}, \quad \overline{x_i} \vee \overline{x_j} \vee \overline{x_k}.$
- If b is -1, then we add the following clauses:
  $\overline{x_i} \vee \overline{x_j} \vee x_k, \quad x_i \vee \overline{x_j} \vee \overline{x_k}, \quad \overline{x_i} \vee x_j \vee \overline{x_k}, \quad x_i \vee x_j \vee x_k.$

Note that ϕ contains $4m$ clauses. Let $a \in \{1,-1\}^X$ be an assignment. Consider any equation e in L.

- If e is satisfied by a, then all 4 clauses in ϕ corresponding to e are satisfied by a.
- If e is not satisfied by a, then precisely 3 of the 4 clauses are satisfied by a.

Using this observation Håstad proves that these instances have a gap of $(1-\frac{3}{4}\delta, \frac{7}{8}+\frac{1}{8}\delta)$. We observe that in the proof, Håstad constructs a witness transformation for the reduction.

Lemma 4.54. *Let β be the reduction described in 4.53. There exists a witness transformation for β that transforms an assignment satisfying at least $(1-\delta)$ fraction of equations in a system of linear equations L into an assignment that satisfies at least a $(1-\frac{3}{4}\delta)$ fraction of clauses of $\beta(L)$.*

Proof. Let L be a $(1-\delta)$-satisfiable system of linear equations and let $\phi = \beta(L)$. Let $a \in \{1,-1\}^X$ be an assignment that satisfies at least $(1-\delta)m$ equations of L. This assignment satisfies at least $4m - 3m\delta = (1-\frac{3}{4}\delta)4m$ clauses of ϕ. So ϕ is at least $(1-\frac{3}{4}\delta)$-satisfiable. □

Lemma 4.55. *If L is at most $(\frac{1+\delta}{2})$-satisfiable, then ϕ is at most $(\frac{7}{8}+\frac{1}{8}\delta)$-satisfiable.*

Proof. Let $a \in \{1,-1\}^X$ be an assignment that satisfies at most $(\frac{1+\delta}{2})m$ equations of L. This assignment satisfies at most $4m(\frac{1+\delta}{2}) + 3m(\frac{1-\delta}{2}) = (\frac{7}{8}+\frac{1}{8}\delta)4m$ clauses of ϕ. So ϕ is at most $(\frac{7}{8}+\frac{1}{8}\delta)$-satisfiable. □

□

By the same argument as corollary 4.51.1, Håstad shows that the $\frac{7}{8}$-approximation algorithm for MAX-3SAT is optimal.

Corollary 4.53.1. *For any $\epsilon > 0$, there exists no polynomial-time $(\frac{7}{8} + \epsilon)$-approximation algorithm for MAX-3SAT, unless $\boldsymbol{P = NP}$.*

Let L be a language in **NP** and let $x \in \{0,1\}^*$ be an instance. By the same reasoning from corollary 4.51.1, we can reduce x to a system of linear equations $L = L_x$ and by theorem 4.53 we can reduce this system to a 3CNF formula $\phi = \phi_L$ that is either at least $(1-\epsilon)$-satisfiable or at most $\frac{7}{8} + \epsilon$. A $\frac{7}{8} + \epsilon$ can be used to distinguish between these instances and thus decide whether $x \in L$. Unless $\mathbf{P} = \mathbf{NP}$, this is not possible in polynomial time.

5 Proof of claim 3.2 and 3.3

We are now in the position to complete the proof of theorem 3.1 by proving claim 3.2 and 3.3.

Claim 3.2. There exists a Karp reduction α from factoring to 3SAT-N with the following property.

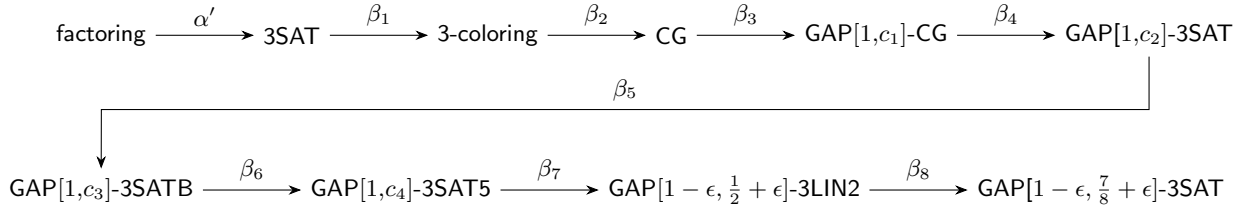
1. Let $N := (n,k) \in$ factoring be a factoring instance and let d be a divisor of n with $d \leq k$. By the definition of a Karp reduction it holds that $(\phi, N) := \alpha(N) \in$ 3SAT-N. On input N and d, we can in polynomial time in $|N| + |d|$ construct a satisfying assignment for ϕ.

Proof. Let α' be the Karp reduction as laid out in theorem 4.1 and let α map a factoring instance N to the pair $(\alpha'(N), N)$. By lemma 4.3, property 1 holds. □

Claim 3.3. For any sufficiently small $\epsilon > 0$, there exists a gap-producing reduction $\beta := \beta_\epsilon$ from 3SAT-N to GAP$[1-\epsilon, \frac{7}{8}+\epsilon]$-3SAT-N with the following properties.

1. Let $(\phi, N) \in$ 3SAT-N and let a be a satisfying assignment for ϕ. By the definition of a gap-producing reduction it holds that $(\psi, N') := \beta(\phi, N) \in L_{yes}$. On input ϕ and a, we can in polynomial time in $|\phi| + |a|$ construct an assignment for ψ that satisfies a $1-\delta$ fraction of its clauses.
2. Let (ϕ, N) be a GAP$[1-\delta, \frac{7}{8}+\delta]$-3SAT-N instance. Given (ϕ, N), we can, in polynomial time in the size of (ϕ, N), decide whether there exists a factoring instance (N') such that $(\phi, N) = \beta(\alpha(N'))$ and if it exists, we can compute it in polynomial time.

Proof. Let β_1, β_2, β_3, β_4, β_5, β_6, $\beta_7 := \beta_{7,\epsilon}$ and $\beta_8 := \beta_{8,\epsilon}$ be the reductions from lemma 4.7, lemma 4.9, theorem 4.6, corollary 4.5, proposition 4.35, proposition 4.32, proposition 4.51 and theorem 4.53 respectively. A visualization of all reductions can be seen below.



Define β' to be the chain of these reductions, applying β_1 first and β_8 last:

$$\beta' := \beta_8 \circ \ldots \circ \beta_1$$

We define $\beta(\phi, N) := (\beta'(\phi), N)$ for a 3SAT-N instance (ϕ, N). Since β' is the composition of polynomial-time reductions, β can be computed in polynomial time.

By the accompanying lemmas there exists a witness transformation for each reduction β_i. By lemma 3.4, the composition of these transformations is again a witness transformation. This transformation outputs an assignment for the formula resulting from β' that satisfies at least a $1-\epsilon$ fraction of clauses. So, property 1 holds.

Let (ϕ, N) be a GAP$[1-\delta, \frac{7}{8}+\delta]$-3SAT-N instance. We can simply read out N and compute $\beta(\alpha(N))$ to see if it is the pre-image of (ϕ, N). So, property 2 holds. □

6 Expansion of the main theorem

The Cook-Levin theorem allows us to reduce any **NP** problem to 3SAT and thus to GAP$[1-\epsilon, \frac{7}{8}+\epsilon]$-3SAT. We introduce a different **NP** problem that can be used as an alternative for the factoring problem.

Definition 6.1. Let G be a cyclic group and let $h, g \in G$ be elements in this group. A positive integer k is called a discrete logarithm of h with base g, denoted $k := \log_g(h)$ if $g^k = h$.

Let $(g, h) \in G$ and let $l \in \mathbb{N}$. We consider the decision problem of deciding whether there exists a discrete logarithm less than some l. It is widely believed that this problem is not in **P**. This problem is clearly in **NP**, because we can verify for some candidate witness $k \in \mathbb{N}$ whether $k \leq l$ and whether k is the discrete logarithm of h with base g in polynomial time. Due to Shor, we can in polynomial time compute the discrete logarithm of h with base g, so we can find a witness in polynomial time.

In the proof of the main theorem we assume that factoring $\notin$ **P** and use the fact that factoring $\in$ **NP** and a witness can be found in polynomial time using a quantum Turing machine. We do not rely on any other properties of the factoring problem. So, the proof of main theorem is still valid if we assume that the decision version of the discrete logarithm problem is not in **P**and interchange it with the problem of integer factoring.

Generally, instead of factoring we can base the proof on any problem with the correct requirements. Instead of assuming that factoring $\notin$ **P**, we assume that there exists any **NP** decision problem for which we can find a witness using a polynomial time quantum Turing machine, but that is not decidable in polynomial on a classical Turing machine. Note that the condition that we can find a witness is stronger than the condition that the problem is in **BQP**. The arguments of theorem 3.1 still hold when we use such a problem rather than factoring.

References

[1] Scirate discussion page for arxiv:2212.12572. `https://scirate.com/arxiv/2212.12572`. Accessed: 2026-06-03.

[2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of Mathematics*, 160(2):781–793, 2004.

[3] M. Ajtai. Recursive construction for 3-regular expanders. In *Combinatorica 14*, page 379–416, 1994.

[4] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

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

[6] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[7] James A. Carlson, Arthur Jaffe, and Andrew Wiles, editors. *The Millennium Prize Problems*. American Mathematical Society and Clay Mathematics Institute, Providence, RI, 2006.

[8] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158. ACM, 1971.

[9] Irit Dinur. The pcp theorem by gap amplification. *J. ACM*, 54(3):12–es, June 2007.

[10] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.

[11] Oded Goldreich. *On promise problems: a survey*, page 254–290. Springer-Verlag, Berlin, Heidelberg, 2006.

[12] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, jul 2001.

[13] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103, New York, 1972. Plenum Press.

[14] Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[15] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2(1), January 2016.

[16] Christos Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, page 229–234, New York, NY, USA, 1988. Association for Computing Machinery.

[17] Niklas Pirnay, Vincent Ulitzsch, Frederik Wilde, Jens Eisert, and Jean-Pierre Seifert. An in-principle super-polynomial quantum advantage for approximating combinatorial optimization problems via computational learning theory. *Science Advances*, 10(11):eadj5170, 2024.

[18] Ran Raz. A parallel repetition theorem. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, page 447–456, New York, NY, USA, 1995. Association for Computing Machinery.

[19] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[20] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[21] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 3rd edition, 2013.

[22] Mario Szegedy. Quantum advantage for combinatorial optimization problems, simplified, 2022.

[23] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1936.