Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Quantum Approximate Optimisation for Boolean Satisfiability

Supervisor:
Dr. Roberto Bondesan

Author:
Andrew Elkadi

Second Marker:
Dr. Herbert Wiklicky

June 21, 2023

Abstract

Implementing a fault-tolerant quantum computer is a challenging task, made difficult by system-environment interactions and the low temperatures required for stable operation. The current state of quantum computing, the Noisy Intermediate-Scale Quantum (NISQ) era, is characterised by quantum processors with few, error-prone, qubits. As such, they are incapable of supporting circuits that require large depths. This has invited interest in developing NISQ-era algorithms, often making use of hybrid approaches where classical and low-depth quantum processors work in parallel. A popular example, the Quantum Approximate Optimisation Algorithm (QAOA), is used for solving general combinatorial optimisation problems.

In this work, we explore the application of QAOA in solving the Boolean satisfiability problem (SAT), and its variant Not-All-Equal (NAE) SAT. We focus on regimes where the problems are known to have solutions with low probability: the *satisfiability ratio*. While classical algorithms are known to struggle solving problems efficiently near this threshold, recent work has shown a quantum advantage afforded by QAOA. We develop novel encodings of SAT and NAE-SAT for QAOA, allowing us to confirm the quantum advantage for SAT and establish a quantum advantage for NAE-SAT. In doing so, we present new findings on the performance of QAOA for a range of these problems' instances.

Acknowledgements

Firstly, I would like to express my gratitude to Dr. Roberto Bondesan, who has not only been an encouraging supervisor throughout the course of this project but also a supportive mentor over the past year. Additionally, I would like to extend my thanks to Dr. Herbert Wiklickly for his guidance, particularly his invaluable writing advice.

Finally, thank you to my friends and family.

Contents

Chapter 1

Introduction

The Boolean satisfiability problem (SAT) is the canonical NP-complete problem [1]. While this means it is unlikely that a polynomial-time solving algorithm will be found, its wide range of applications invite interest in finding *efficient enough* algorithms [2]. This includes areas of: circuit design [3], logic-based planning [4] and bug detection [5].

An instance of SAT is a Boolean formula consisting of n variables arranged in m clauses. Each clause represents a constraint on the variables, and to solve SAT is to determine whether there exists a variable assignment that satisfies all the constraints. When each clause is restricted to k variables the resulting problem is known as k-SAT [6].

Of particular interest are instances with *clause densities* $r = m/n$ that undergo *computational phase transitions* [6]. The *satisfiability ratio* r_k describes a critical value of r beyond which instances are unlikely to have any solutions. On the other hand, the *algorithmic ratio* a_k is a threshold beyond which no classical algorithm is known to find solutions efficiently. Notably, it is known that the algorithmic ratio is strictly smaller than the satisfiability ratio $(a_k < r_k)$[7].

A natural place to search for algorithms to solve instances with $a_k \leq r \leq r_k$ is quantum computing. In particular, we consider the application of the quantum approximate optimisation algorithm (QAOA) [8]. QAOA is designed to find approximate solutions to general combinatorial optimisation problems. Its low circuit depth makes it an attractive candidate as an algorithm to be run on NISQ-era quantum computers [9]. It is based on the trotterised adiabatic process and is an instance of a Variational Quantum Eigensolver (VQE) [8].

The recent work of Boulebnane & Montanaro [10] explores the *success probability* and *median running time* of QAOA on k-SAT instances, discovering a quantum advantage over classical algorithms. In particular, they benchmark QAOA against a range of classical solvers and find that it outperforms the best performing algorithm, `WalkSATlm` [11]. In this project, we consider these heuristics and study the performance of QAOA on a related problem: k-NAE-SAT. Here, in addition to satisfying all constraints, the variables within a clause cannot all be assigned the same value [12].

1.1 Contributions

In this work, we:

- Derive novel encodings of k-SAT and k-NAE-SAT for QAOA that can be run on any gate-based quantum computer and implement them in `Qiskit` [13]. Studying the cost of classically simulating these, we introduce *diagonalisations* of our Hamiltonians that allow us to reduce the effective time complexity from $\mathcal{O}\big(2^k k^3 N^2 \log N\big)$ to $\mathcal{O}(N)$, for a system size of $N = 2^n$.

- Reproduce the results of Boulebnane & Montanaro [10], implementing an efficient simulator in `PyTorch` [14], and confirm the quantum advantage of QAOA for 8-SAT over the best performing classical solver `WalkSATlm`.
- Produce novel *success probability* and *median running time* results of QAOA for 4-SAT and a range of k-NAE-SAT problems. We also consider the scaling of these heuristics in the large p limit and interpret their *excessive scaling* performance.
- Introduce `WalkSATm2b2`, a solver for k-NAE-SAT, that extends on `WalkSATlm` and accounts for the symmetry of the problem's instances. We show that it outperforms `WalkSATlm` on k-NAE-SAT, yet that QAOA outperforms both solvers on a range of instances.

1.2 Ethical Considerations

This project has no legal or licensing concerns. The libraries being used are all open source and freely available through a standard package manager.

Although solving satisfiability problems has a range of applications, including ones that could indirectly be malicious, these are not direct consequences of work being done in this thesis.

Nevertheless, the consumption of electricity due to the simulations carried out in the course of this project should be considered. While distributing and parallelising processes is powerful, it is easy to overlook the environmental impact of such large scale compute. In particular, such speed ups allowed for more experiments to be carried out than otherwise would have been possible.

Finally, no humans or animals were used in this project (this would require too many qubits to encode anyways).

Chapter 2

Preliminaries

2.1 Boolean Satisfiability Problem

The Boolean satisfiability problem (SAT) is a combinatorial optimisation problem that seeks a *satisfying assignment* for a *Boolean propositional formula* [1]. Simply, given a set of constraints on variables, we ask whether there exists an assignment of values to these variables that satisfies the constraints. This is defined formally in what follows, based on the work of Moore & Mertens [6].

2.1.1 Propositional Formulae

Definition 2.1.1. *(Boolean variable) A Boolean variable x is a variable that can be assigned a **truth value** of either $v(x) = \top$ or $v(x) = \bot$, under the assignment v.*

Alternatively, and often for brevity, we consider Boolean variables as variables ranging over the values $\{0, 1\}$ [15]. A value of 0 corresponds to the truth value $\bot$ and 1 the truth value $\top$. For a collection of variables $\{x_0, x_1, \ldots, x_{n-1}\}$ their truth values can be succinctly written as the bitstring $\underline{x} \in \{0, 1\}^n$.

Definition 2.1.2. *(Boolean literal) A Boolean literal l is a Boolean variable x or its negation, denoted $\neg x$.*

The negation of a Boolean variable is simply another variable that takes the opposing truth value. As such, $v(\neg x) = \top$ iff $v(x) = \bot$.

Definition 2.1.3. *(Propositional formula) A propositional formula ϕ is a collection of Boolean literals $\{l_0, \ldots, l_n\}$, conjunctions $\wedge$ and disjunctions $\vee$.*

The truth value of a Boolean variable in isolation is not very expressive. However, the truth value of a collection of these variables can be used to realise many real-world constraints. The connectives, $\wedge$ and $\vee$, are used to represent situations where we may require combinations of variables to have particular truth values, or mutually exclusive truth values.

Definition 2.1.4. *(Truth value of a propositional formula) Recursively extending the notion of a truth value:*

- $v(l_1 \wedge l_2) = \top$ *iff* $v(l_1) = v(l_2) = \top$
- $v(l_1 \vee l_2) = \top$ *iff* $v(l_1) = \top$ *and/or* $v(l_2) = \top$

Definition 2.1.5. *(Satisfying a propositional formula) A propositional formula ϕ is satisfied by assignment v iff $v(\phi) = \top$.*

In the alternative representation, if the bitstring $\underline{x}$ (corresponding to the assignment v) satisfies ϕ, it is denoted as [15]:

$$\underline{x} \vdash \phi \tag{2.1}$$

Example 2.1.1. *Alice will only go to the shops if she can guarantee she'll stay dry. She'll stay dry if and only if it doesn't rain or the underpass is open. Let r be the Boolean variable such that $v(r) = \top$ if and only if it is raining and p the Boolean variable such that $v(p) = \top$ if and only if the underpass is open. The propositional formula $d = \neg r \vee p$ represents whether Alice will stay dry. If $v(d) = \top$, she will, if $v(d) = \bot$, she will not.*

SAT considers a class of propositional formulas with the added structure of being in Conjunctive Normal Form.

Definition 2.1.6. *(CNF) A propositional formula ϕ is in Conjunctive Normal Form (CNF) if it is a conjunction of m clauses $\bigwedge_{i=0}^{m-1} c_i$, where each clause is a disjunction of k_i Boolean literals $c_i = \bigvee_{j=0}^{k_i-1} l_{ij}$.*

By these definitions, it is clear that a propositional formula in CNF is satisfied if and only if **all** of its clauses are satisfied. A clause on the other hand is satisfied if and only if **any** of its Boolean literals are satisfied. Finally, we define SAT.

Definition 2.1.7. *(Boolean satisfiability problem - SAT) Given a propositional formula ϕ, in CNF, does there exist an assignment v on ϕ such that v satisfies ϕ?*

It is worth noting, any propositional formula can be converted into an equivalent CNF representation. As such, this does not restrict the domain of problems SAT considers. An alternative structure, Disjunctive Normal Form (DNF), which any propositional formula can also be written in, considers a disjunction of clauses which themselves are conjunctions of literals ($\phi' = \bigvee_{i=0}^{m-1} \bigwedge_{j=0}^{k_i-1} l_{ij}$). However, the SAT problem for DNF formulas is in P, meaning this formulation is of lesser interest.

Remark 2.1.1. *While one might consider solving SAT by first converting the propositional formula into DNF, this is not generally possible in polynomial time. In the worst case, the equivalent DNF structure is exponentially greater in size, requiring exponential time to construct. Converting to CNF on the other hand can always be done in polynomial time [6].*

Example 2.1.2. *The propositional formula $(x_1 \vee \neg x_2) \wedge x_3$ is satisfied by the assignment $v(x_1) = v(x_2) = v(x_3) = \top$, amongst others.*

Example 2.1.3. *The propositional formula $x_1 \wedge \neg x_1$ is unsatisfiable. A satisfying assignment would require both x_1 and its negation to be true. This is not possible since $v(\neg x_1) = \neg v(x_1)$.*

Example 2.1.4. *The propositional formula $(x_1 \vee y_1) \wedge (x_2 \vee y_2)$ converted to DNF takes the form:*

$$\begin{aligned}(x_1 \vee y_1) \wedge (x_2 \vee y_2) &\equiv (x_1 \wedge (x_2 \vee y_2)) \vee (y_1 \wedge (x_2 \vee y_2)) \\ &\equiv (x_1 \wedge x_2 \vee x_1 \wedge y_2) \vee (y_1 \wedge x_2 \vee y_1 \wedge y_2) \\ &\equiv (x_1 \wedge x_2) \vee (x_1 \wedge y_2) \vee (y_1 \wedge x_2) \vee (y_1 \wedge y_2)\end{aligned} \tag{2.2}$$

where we have used De Morgan's laws to expand the terms [6]. This has $2^3 = 8$ literals.

Example 2.1.5. *In general, the propositional formula $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \cdots \wedge (x_n \vee y_n)$ requires 2^{n+1} literals when converted to DNF [6].*

2.1.2 k-SAT and k-NAE-SAT

SAT is an NP-complete problem [1], meaning it is in NP and that any NP problem can be reduced to it in polynomial time [6]. As such, many problems of interest can be solved by first representing them as propositional formulae and then determining whether a satisfying assignment exists [2]. Similarly, by transitivity, showing that SAT reduces to an NP-hard problem allows the problem to be classified as NP-complete. In particular cases, variants of SAT - that reduce to/from SAT - are useful to consider as an intermediate reduction. A simple example of this is k-SAT, where there is a constraint on the size of each clause [6].

Definition 2.1.8. *(k-SAT) Given a propositional formula ϕ in CNF, where each clause has k literals, does there exist an assignment v on ϕ such that v satisfies ϕ?*

It is clear that k-SAT is an instance of SAT where each clause takes the form $c_i = \bigvee_{j=0}^{k-1} l_{ij}$.

Another example, and the focus of the latter part of this work, k-NAE-SAT, places a constraint on the literals within a clause having the same truth value.

Definition 2.1.9. *(k-NAE-SAT) Given a propositional formula ϕ in CNF, where each clause has k literals, does there exist an assignment v on ϕ such that v satisfies ϕ and v does not assign all literals within a clause to the same truth value?*

Example 2.1.6. *In the 2-NAE-SAT formulation, the propositional formula $(x_1 \vee \neg x_2) \wedge (x_3 \vee x_4)$ is not satisfied by the assignment $v(x_1) = v(x_3) = \top, v(x_2) = v(x_4) = \bot$. It assigns both literals in the first clause to the same truth value: $v(x_1) = v(\neg x_2) = \top$.*

In other words, k-NAE-SAT requires at least one literal to be true, and at least one literal to be false, in each clause. Unlike k-SAT, k-NAE-SAT is symmetric with respect to flipping the assignment values. This makes it especially useful when considering problems with a similar symmetry [6]. An example of this is the Graph-k-Colouring problem. It asks whether the nodes in a graph can be coloured with k colours such that no two connected nodes are the same colour. k-NAE-SAT can be reduced to Graph-k-Colouring by representing the variables in a clause as a collection of connected nodes. This is done in such a way that the nodes being colourable corresponds to the clause being satisfiable. It is clear that this will only hold if the collection of nodes are not coloured the same colour - equivalently, the literals in the clause are not all assigned the same truth value [6].

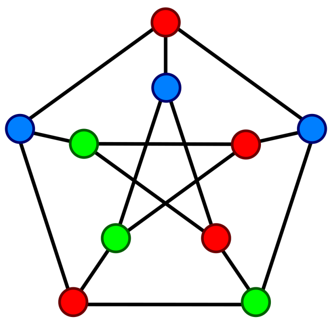


Figure 2.1: Valid 3-Colouring of the 'Petersen Graph' [16].

2.1.3 Computational Phase Transitions

In this work, we consider solving randomly generated k-SAT and k-NAE-SAT problems. For each, given a CNF formula of n variables and m clauses, we are interested in two quantities: the *satisfiability ratio* and *algorithmic ratio*. We begin by making precise the notion of a random problem before defining these quantities.

Definition 2.1.10. *(Random CNF instance) A random CNF instance of n variables and m clauses is denoted $F_k(n, m)$. For each clause, **with replacement**, k variables are chosen with uniform randomness from $\binom{n}{k}$ combinations. Each variable is randomly negated with probability $p = 0.5$ [6].*

We are interested in regimes where $F_k(n, m)$ instances are unsatisfiable and describe these using the notion of *clause density*.

Definition 2.1.11. *(Clause density) $F_k(n, m)$ has clause density $r = \frac{m}{n}$ [6].*

Experimental evidence suggests that there exists a threshold α_c, above which instances with clause densities $r > \alpha_c$ become unsatisfiable. Figure 2.2 explores this for 3-SAT and shows that unsatisfiability appears at $\alpha_c \approx 4.267$. While this change occurs continuously, it becomes sharper as n increases, suggesting that in the limit $n \to \infty$ it becomes a discontinuous jump [6]. Formally, this discontinuous jump is defined using the notion of *satisfiability with high probability*.

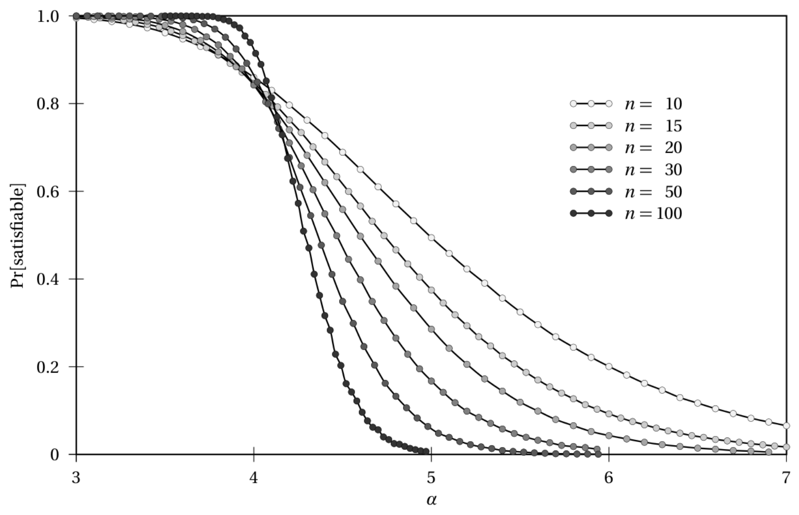


Figure 2.2: The probability that an $F_3(n, m)$ 3-SAT instance is satisfiable as a function of the clause density α, for various values of n. Sample size varies from 10^6 for $n = 10$ to 10^4 for $n = 100$ [6].

Definition 2.1.12. *(With High Probability) $F_k(n, m)$ is satisfiable/unsatisfiable 'with high probability' (w.h.p) if [6]*

$$\lim_{n\to\infty} P[F_k(n, m) \textit{ satisfiable/unsatisfiable}] = 1 \tag{2.3}$$

Definition 2.1.13. *(Satisfiability Ratio) The satisfiability ratio, $r_k(n)$, is a sharp threshold on clause density r such that $F_k(n, m)$ instances are [6]:*

$$\begin{aligned} \textit{Satisfiable w.h.p } \forall \varepsilon > 0 : r < r_k(n) - \varepsilon \\ \textit{Unsatisfiable w.h.p } \forall \varepsilon > 0 : r > r_k(n) + \varepsilon \end{aligned} \tag{2.4}$$

Although $r_k(n)$ is not known precisely for $k \geq 3$, it has been shown that [7]:

$$\begin{aligned} 2^k \ln(2) - \mathcal{O}(k) \leq r_k(n) \leq 2^k \ln(2), \text{ for } k\text{-SAT} \\ 2^{k-1} \ln(2) - \mathcal{O}(1) \leq r_k(n) \leq 2^{k-1} \ln(2), \text{ for } k\text{-NAE-SAT} \end{aligned} \tag{2.5}$$

While $r_k(n)$ is dependent on n, it is conjectured to converge in the limit for fixed k [7].

Besides the satisfiability ratio, we are interested in problems that are satisfiable but cannot be solved *efficiently* with currently known classical algorithms.

Definition 2.1.14. *(Algorithmic Ratio) The algorithmic ratio, a_k, is a sharp threshold on clause densities r such that there are no efficient (polynomial time) known classical solvers for $F_k(n, m)$ instances [7].*

No classical algorithm is known to find solutions efficiently beyond [7]:

$$\begin{aligned} a_k &= \frac{2^k}{k} \ln(k), \text{ for } k\text{-SAT} \\ a_k &= \frac{2^{k-1}}{k} \ln(k), \text{ for } k\text{-NAE-SAT} \end{aligned} \tag{2.6}$$

Combining 2.5 and 2.6, for k large, we draw our attention to the relationships:

$$
\begin{aligned}
\frac{r_k(n)}{a_k} &\geq \frac{k\ln(2)}{\ln(k)} - \frac{\mathcal{O}(k)}{2^k\ln(k)} \geq \frac{k\ln(2)}{\ln(k)} - 1 > 1, \text{ for } k\text{-SAT} \\
\frac{r_k(n)}{a_k} &\geq \frac{k\ln(2)}{\ln(k)} - \frac{\mathcal{O}(1)}{2^{k-1}\ln(k)} \geq \frac{k\ln(2)}{\ln(k)} - 1 > 1, \text{ for } k\text{-NAE-SAT}
\end{aligned}
\tag{2.7}
$$

This implies that there exists problems in between the satisfiability and algorithmic ratios. We aim to explore QAOA's use in solving problems within this regime.

2.2 Classical Solvers

Classical algorithms for solving SAT can be split into two main categories: Complete algorithms and Stochastic Local Search (SLS) algorithms [11].

SLS algorithms are *incomplete* in the sense that they cannot determine with certainty whether a given formula is satisfiable. However, in the regime of necessarily satisfiable problems, they are very efficient, particularly for randomly generated instances. Their general approach consists of starting with a random assignment and flipping variable truth values until a satisfying assignment is found. Deciding which variables to flip is done through a combination of randomness or *locally searching* the space of truth assignments to optimise a constructed heuristic.

2.2.1 DPLL

Davis, Putnam, Logemann, and Loveland (DPLL) is a standard example of a Complete algorithm [17]. Its approach (Algorithm 1) is to choose a variable x, assign the two possible truth values to it and recursively check if either of the two resulting settings is satisfiable. Setting $v(x) = \top$ satisfies all the clauses containing x, while all the clauses containing $\neg x$ must now be satisfied by another variable within it. In the former, we can think of the formula shortening and in the latter the clause shortening. If a clause becomes empty (i.e. all variables are given assignments that disagree with the clause), then we have a contradiction and DPLL must backtrack. Finally, if every computation leaf has a contradiction then the formula is *certainly* unsatisfiable.

It is clear that DPLL's run time corresponds to the number of recursive calls made (nodes searched). As such, an increase in backtracking corresponds to an increase in run time. When the clause density is low, little backtracking is observed since each clause shares few variables with others. As a result, the impact of setting a variable's truth value in one clause is unlikely to cause a contradiction in another and require a backtrack. However, as the density increases, so does the overlap in variables between clauses, causing more backtracking and recursive calls. Eventually, this becomes an exponential amount [6].

2.2.2 WalkSAT and WalkSATlm

In general, SLS algorithms select variables to flip based on optimising *scoring functions*.

Definition 2.2.1. *(Make score) For a variable x in a Boolean formula ϕ, $make(x)$ is the number of clauses that become* ***satisfied*** *by flipping x.*

Definition 2.2.2. *(Break score) For a variable x in a Boolean formula ϕ, $break(x)$ is the number of clauses that become* ***unsatisfied*** *by flipping x.*

An example of a maximised scoring function is

$$score(x) = make(x) - break(x) \tag{2.8}$$

Algorithm 1: DPLL Algorithm

```
1  Function DPLL($\phi$):
2      if $\phi$ empty then
3          return true;
4      if $\phi$ contains an empty clause then
5          return false;
6      Choose an unset variable $x$ from $\phi$;
7      if DPLL($\phi[v(x) = \top]$) then
8          return true;
9      if DPLL($\phi[v(x) = \bot]$) then
10         return true;
11     return false;
```

A *focused random walk* is an SLS approach that chooses variables only from clauses that are unsatisfied. `WalkSAT` [18], is a popular example that employs a variable selection scheme (Algorithm 2) based on *break* scores. At each iteration, any *freebie* variables are flipped (variables with *break* scores of 0). Otherwise, a Bernoulli random variable (controlled by *noise* parameter p) is sampled. Depending on the outcome, a variable to be flipped is either chosen at random or amongst those that have lowest (non-zero) *break* scores in the clause. `WalkSAT` solves *tiebreaks* (multiple variables with optimal score) through random selection. We note the inclusion of a *max_flips* argument to account for SLS being incomplete, avoiding the algorithm looping infinitely in the case of an unsatisfiable formula.

`WalkSATlm` [11], builds on `WalkSAT`, altering the tiebreak procedure to no longer be random. A new scoring function is introduced as follows.

Definition 2.2.3. *(τ-satisfied) Given a Boolean formula ϕ and assignment α, a clause c in ϕ is τ-satisfied iff under α, it contains exactly τ satisfied literals [11].*

As such, a 0-satisfied clause is an unsatisfied clause, while $\geq$ 1-satisfied clauses are satisfied. 1-satisfied clauses are of interest as they are very unstable - flipping the variable corresponding to the satisfied literal returns the clause to an unsatisfied state. Using this, the *make* score is generalised.

Definition 2.2.4. *(τ-level make) For a variable x in a Boolean formula ϕ, $make_\tau(x)$ is the number of $(\tau - 1)$-satisfied clauses that become τ-satisfied by flipping x [11].*

It is clear that $make_1$ is equivalent to *make*. $make_2$, whereas, is the number of clauses that move away from instability - flipping x would make them 2-satisfied and so they can no longer be broken by flipping exactly one variable. This corresponds to the number of variables that would have their *break* scores decreased by flipping x.

`WalkSATlm` introduces the *lmake* scoring function to be used in tiebreak situations (Algorithm 3).

$$lmake(x) = \omega_1 \cdot make_1(x) + \omega_2 \cdot make_2(x) \tag{2.9}$$

where ω_1, ω_2 are hyperparameters. The variable with the maximal *lmake* score amongst those with the minimal *break* scores is chosen to be flipped. This simple change allows `WalkSATlm` to outperform `WalkSAT` by orders of magnitude for $k > 3$ and stems from the fact that tiebreaks occur in 40% and 30% of steps in 5-SAT and 7-SAT respectively [11].

Algorithm 2: WalkSAT Algorithm

```
Function WalkSAT($\phi$, $p$, $max\_flips$):
    Randomly assign truth values to all variables in $\phi$;
    for $i = 1$ to $max\_flips$ do
        if $\phi$ is satisfied then
            return the assignment;
        Choose uniformly at random an unsatisfied clause $c$ from $\phi$;
        if $\exists x \in c : break(x) = 0$ then
            Flip the value of (the first such) $x$;                    // 'Freebie'
        Sample $X \sim Bernoulli(p)$;
        if $X$ then
            Choose uniformly at random $x \in c$ and flip its value;
        else
            Choose uniformly at random $x \in \operatorname{argmin}_{y \in c} break(y)$ and flip its value;
    return No satisfying assignment found;
```

Algorithm 3: WalkSATlm Algorithm

```
Function WalkSATlm($\phi$, $p$, $max\_flips$):
    Randomly assign truth values to all variables in $\phi$;
    for $i = 1$ to $max\_flips$ do
        // As in WalkSAT
        Sample $X \sim Bernoulli(p)$;
        if $X$ then
            Choose uniformly at random $x \in c$ and flip its value;
        else
            Choose uniformly at random $x \in \operatorname{argmax}_{v \in B} lmake(v)$, where $B = \operatorname{argmin}_{y \in c} break(y)$,
            and flip its value;
    return No satisfying assignment found;
```

2.3 Quantum Approximate Optimisation Algorithm

2.3.1 Adiabatic Quantum Computing

The origins of QAOA lie in Adiabatic Quantum Computing (AQC), also known as Quantum Annealing [19].

The evolution of a quantum system under a time dependent Hamiltonian $\hat{\mathcal{H}}(t)$ is governed by Schrödinger's equation [20]:

$$i\frac{d}{dt}|\psi(t)\rangle = \hat{\mathcal{H}}(t)|\psi(t)\rangle \tag{2.10}$$

(where $\hbar$ has been set to 1).

Theorem 2.3.1. *A physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum [21].*

Mathematically, define the instantaneous eigenstates and eigenvalues of $\hat{\mathcal{H}}(t=0) \in \mathbb{C}^{N\times N}$ by

$$\hat{\mathcal{H}}(t=0)\,|\lambda, t=0\rangle = E_\lambda(t=0)\,|\lambda, t=0\rangle \tag{2.11}$$

with $N = 2^n$ for some $n \in \mathbb{N}$, $E_0(t=0) \le E_1(t=0) \le \dots \le E_{N-1}(t=0)$.

Let $|\psi(t=0)\rangle$ an instantaneous eigenstate of $\hat{\mathcal{H}}(t=0)$ for some $\lambda \in \{0, \dots, N-1\}$, i.e.

$$\hat{\mathcal{H}}(t=0)\,|\psi(t=0)\rangle = E_\lambda(t=0)\,|\psi(t=0)\rangle \tag{2.12}$$

If $\forall t : 0 \le t \le T$

$$E_{\lambda'}(t) - E_\lambda(t) \begin{cases} > 0 & \lambda' > \lambda \\ < 0 & \lambda' < \lambda \end{cases} \tag{2.13}$$

then [19]

$$\lim_{T\to\infty} |\,\langle \lambda, s=T|\psi(t=T)\rangle\,| = 1 \tag{2.14}$$

Consider an *interpolating* Hamiltonian

$$\hat{\mathcal{H}}_{\mathcal{A}}(t) = f(t)\hat{\mathcal{H}}_{\mathcal{I}} + g(t)\hat{\mathcal{H}}_{\mathcal{F}} \tag{2.15}$$

between some *initial* Hamiltonian $\hat{\mathcal{H}}_{\mathcal{I}}$ and *final* Hamiltonian $\hat{\mathcal{H}}_{\mathcal{F}}$, with boundary conditions

$$\begin{aligned} \hat{\mathcal{H}}_{\mathcal{A}}(0) = \hat{\mathcal{H}}_{\mathcal{I}} &\Rightarrow f(0)=1, g(0)=0 \\ \hat{\mathcal{H}}_{\mathcal{A}}(T) = \hat{\mathcal{H}}_{\mathcal{F}} &\Rightarrow f(T)=0, g(T)=1 \end{aligned} \tag{2.16}$$

By application of 2.3.1, preparing the system to start in the λ^{th} eigenstate of $\hat{\mathcal{H}}_{\mathcal{I}}$ and evolving for sufficiently large T will leave us in the λ^{th} eigenstate of $\hat{\mathcal{H}}_{\mathcal{F}}$. If finding the λ^{th} eigenstate of $\hat{\mathcal{H}}_{\mathcal{F}}$ directly is difficult, then this provides an alternative method of calculating it.

Remark 2.3.1. *$\hat{\mathcal{H}}_{\mathcal{A}}$ usually takes the form*

$$\hat{\mathcal{H}}_{\mathcal{A}} = \left(1 - \frac{t}{T}\right)\hat{\mathcal{H}}_{\mathcal{I}} + \frac{t}{T}\hat{\mathcal{H}}_{\mathcal{F}} \tag{2.17}$$

but any smooth f, g satisfying the boundary conditions are sufficient.

The necessary run time of the algorithm typically scales as [22]

$$T = O\left(\Delta_{min}^{-2}\right) \tag{2.18}$$

where

$$\begin{aligned} \Delta_{min} &= \min\left(\delta_-(\lambda), \delta_+(\lambda)\right) \\ \delta_- &= \min_{0\le t\le T} E_\lambda - E_{\lambda-1} \\ \delta_+ &= \min_{0\le t\le T} E_{\lambda+1} - E_\lambda \end{aligned} \tag{2.19}$$

which reduces to $\Delta_{min} = \delta_-(\lambda)$ for $\lambda = N-1$ and $\Delta_{min} = \delta_+(\lambda)$ for $\lambda = 0$.

However, this scaling means that in certain instances the required run time is

$$T = o(\sqrt{N}) = o(2^{\frac{n}{2}}) \tag{2.20}$$

where we recall that $N = 2^n$. In other words, the running time scales exponentially with the problem size and, as such, the algorithm can be inefficient [23].

2.3.2 Trotterisation

Solving equation 2.10 gives rise to the unitary *time evolution operator*

$$\hat{\mathcal{U}}(t) = \mathcal{T}e^{-i\int_0^t \hat{\mathcal{H}}(t')dt'} \tag{2.21}$$

wherein $\mathcal{T}$ is the time ordering operator, that takes any product of operators and reorders them so that each operator only has later operators to the left, and

$$|\psi(t)\rangle = \hat{\mathcal{U}}(t)\,|\psi(0)\rangle \tag{2.22}$$

where $|\psi(0)\rangle$ is the system's initial state [20].

We are interested in accurate, yet practical, approximations of this operator. As such, we consider the corresponding Riemann sum to remove the need for $\mathcal{T}$:

$$\hat{\mathcal{U}}(t) \approx e^{-i\sum\limits_{a=0}^{k-1}\tau\hat{\mathcal{H}}(a\tau)} \tag{2.23}$$

where the elapsed time t has been discretised into k intervals of length $\tau = t/k$. Next, we apply the Trotter-Suzuki Decomposition to rewrite the unitary as a product of operators.

Theorem 2.3.2. *(Trotter-Suzuki Decomposition)*

$$e^{x(A+B)} = e^{Ax}e^{Bx} + \mathcal{O}\big(x^2\big) \tag{2.24}$$

where x is a parameter and A, B are arbitrary operators with $[A,B] \neq 0$ [24].

Therefore, for small enough τ [25]:

$$\hat{\mathcal{U}}(t) \approx \prod_{a=0}^{k-1} e^{-i\hat{\mathcal{H}}(a\tau)\tau} \tag{2.25}$$

For a system governed by the Hamiltonian described in 2.15

$$\hat{\mathcal{U}}_{\mathcal{A}}(t) \approx \prod_{a=0}^{k-1} e^{-i\hat{\mathcal{H}}_{\mathcal{A}}(a\tau)\tau} = \prod_{a=0}^{k-1} e^{-i\left(f(a\tau)\hat{\mathcal{H}}_{\mathcal{I}}+g(a\tau)\hat{\mathcal{H}}_{\mathcal{F}}\right)\tau} \tag{2.26}$$

By a second application of the decomposition (2.3.2)

$$\hat{\mathcal{U}}_{\mathcal{A}}(t) \approx \prod_{a=0}^{k-1} e^{-i\tau f(a\tau)\hat{\mathcal{H}}_{\mathcal{I}}}e^{-i\tau g(a\tau)\hat{\mathcal{H}}_{\mathcal{F}}} := \hat{\mathcal{U}}_{\mathcal{A}}(t;\tau) \tag{2.27}$$

This approximation is exact in the limit

$$\hat{\mathcal{U}}_{\mathcal{A}}(t) = \lim_{\tau\to 0}\hat{\mathcal{U}}_{\mathcal{A}}(t;\tau) \tag{2.28}$$

$\hat{\mathcal{U}}_{\mathcal{A}}(t;\tau)$ is denoted the *trotterised* form of $\hat{\mathcal{U}}_{\mathcal{A}}(t)$.

2.3.3 Variational Quantum Eigensolvers

The Variational Quantum Eigensolver (VQE) [26] is a quantum algorithm used to find the ground state (eigenstate with lowest corresponding eigenvalue) of a physical system governed by Hamiltonian $\hat{\mathcal{H}}$. Solving such a problem has many uses in quantum chemistry and condensed matter physics [27].

Theorem 2.3.3. *(Variational Principle) Let $|\lambda_0\rangle$ the ground state of $\hat{\mathcal{H}}$ such that $\hat{\mathcal{H}}\,|\lambda_0\rangle = \lambda_0\,|\lambda_0\rangle$. The expectation of $\hat{\mathcal{H}}$, for arbitrary state $|\phi\rangle$*

$$\langle\phi|\hat{\mathcal{H}}|\phi\rangle \geq \langle\lambda_0|\hat{\mathcal{H}}|\lambda_0\rangle = \lambda_0 \tag{2.29}$$

Proof. Let $|\phi\rangle$ an arbitrary state. Assume w.l.o.g that it is normalised. The eigenstates of $\hat{\mathcal{H}}$ form a basis of the N-dimensional Hilbert space, allowing us to decompose

$$|\phi\rangle = \sum_{i=0}^{N-1} c_i |\lambda_i\rangle \tag{2.30}$$

where $\hat{\mathcal{H}} |\lambda_i\rangle = \lambda_i |\lambda_i\rangle$. Now

$$\langle\phi|\hat{\mathcal{H}}|\phi\rangle = \sum_{i,j=0}^{N-1} c_i c_j^* \lambda_i \langle\lambda_j|\lambda_i\rangle = \sum_{i=0}^{N-1} |c_i|^2 \lambda_i \geq \lambda_0 \sum_{i=0}^{N-1} |c_i|^2 = \lambda_0 \tag{2.31}$$

by $|\phi\rangle$ normalised. □

VQE makes use of this lower bound by recasting the search for the ground state as a minimisation problem. It prepares a state using a parameterised quantum circuit (PQC), where the unitary $U(\vec{\theta})$ represents its action

$$|\psi(\vec{\theta})\rangle = U(\vec{\theta}) |0\rangle \tag{2.32}$$

The goal is to find

$$\vec{\theta}^* = \underset{\vec{\theta}}{\operatorname{argmin}} \langle\psi(\vec{\theta})|\hat{\mathcal{H}}|\psi(\vec{\theta})\rangle \tag{2.33}$$

Where the state $|\psi(\vec{\theta}^*)\rangle$ is (close to) the ground state.

2.3.4 QAOA Procedure

Inspired by the above, QAOA aims to maximise an objective function

$$\mathcal{C} : \{0,1\}^n \to \mathbb{R} \tag{2.34}$$

where $\{0,1\}^n$ is the n-dimensional vector space of bitstrings.

Definition 2.3.1. *(Hamiltonian representation of a function) A Hamiltonian $\hat{\mathcal{H}}_f$ represents a function f if:*

$$\hat{\mathcal{H}}_f |\underline{x}\rangle = f(\underline{x}) |\underline{x}\rangle \tag{2.35}$$

for $\underline{x} \in \{0,1\}^n$ with corresponding computational basis state $|\underline{x}\rangle$ [28].

If we represent $\mathcal{C}$ by some $\hat{\mathcal{H}}_\mathcal{C}$ then finding

$$\max_{\underline{x}\in\{0,1\}^n} \mathcal{C}(\underline{x}) \tag{2.36}$$

corresponds to finding

$$\max_{\underline{x}\in\{0,1\}^n} \langle\underline{x}|\hat{\mathcal{H}}_\mathcal{C}|\underline{x}\rangle \tag{2.37}$$

by $\langle\underline{x}|\hat{\mathcal{H}}_\mathcal{C}|\underline{x}\rangle = \langle\underline{x}|\mathcal{C}(\underline{x})|\underline{x}\rangle = \mathcal{C}(\underline{x})\langle\underline{x}|\underline{x}\rangle = \mathcal{C}(\underline{x})$.

In other words, in order to maximise the objective function, we seek the highest energy state of $\hat{\mathcal{H}}_\mathcal{C}$. Drawing inspiration from AQC (2.3.1), we consider the effect of initialising our system in the highest energy eigenstate of some simple Hamiltonian $\hat{\mathcal{H}}_\mathcal{B}$ and evolving *slowly* such that the system is finally governed by $\hat{\mathcal{H}}_\mathcal{C}$. QAOA approximates this adiabatic process by trotterising the unitary evolution (2.3.2) and applying VQE (2.3.3) principles (where we note that finding the highest energy state of $\hat{\mathcal{H}}_\mathcal{C}$ is equivalent to finding the ground state of $-\hat{\mathcal{H}}_\mathcal{C}$).

Consider a *mixer* Hamiltonian $\hat{\mathcal{H}}_{\mathcal{B}}$ and *problem* Hamiltonian $\hat{\mathcal{H}}_{\mathcal{C}}$ that *represents* the objective function to maximise. Let

$$\hat{\mathcal{H}}(t) = f(t)\hat{\mathcal{H}}_{\mathcal{B}} + g(t)\hat{\mathcal{H}}_{\mathcal{C}} \tag{2.38}$$

with boundary conditions

$$\begin{aligned} \hat{\mathcal{H}}(0) = \hat{\mathcal{H}}_{\mathcal{B}} \Rightarrow f(0) = 1, g(0) = 0 \\ \hat{\mathcal{H}}(T) = \hat{\mathcal{H}}_{\mathcal{C}} \Rightarrow f(T) = 0, g(T) = 1 \end{aligned} \tag{2.39}$$

The corresponding unitary evolution operator is trotterised (2.27) as:

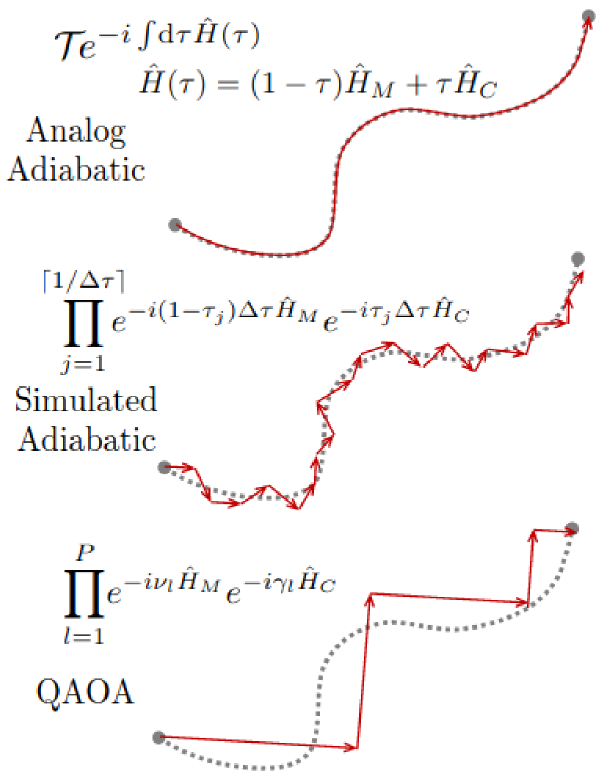
$$\hat{\mathcal{U}}(t) \approx \prod_{a=0}^{k-1} e^{-i\tau f(a\tau)\hat{\mathcal{H}}_{\mathcal{B}}} e^{-i\tau g(a\tau)\hat{\mathcal{H}}_{\mathcal{C}}} \tag{2.40}$$

then *discretised* and *re-parameterised* to form

$$\hat{\mathcal{U}}_{QAOA} := \prod_{i=0}^{p-1} \hat{\mathcal{U}}_{\mathcal{B}}(\beta_i)\hat{\mathcal{U}}_{\mathcal{C}}(\gamma_i) \tag{2.41}$$

for some $p \in \mathbb{N}$, denoted as the *depth* of the QAOA circuit, and unitary operators

$$\begin{aligned} \hat{\mathcal{U}}_{\mathcal{B}}(\beta_j) = e^{-i\beta_j \hat{\mathcal{H}}_{\mathcal{B}}} \\ \hat{\mathcal{U}}_{\mathcal{C}}(\gamma_j) = e^{-i\gamma_j \hat{\mathcal{H}}_{\mathcal{C}}} \end{aligned} \tag{2.42}$$



$\mathcal{T}e^{-i\int d\tau \hat{H}(\tau)}$
$\hat{H}(\tau) = (1-\tau)\hat{H}_M + \tau\hat{H}_C$
Analog Adiabatic
$\prod_{j=1}^{\lceil 1/\Delta\tau \rceil} e^{-i(1-\tau_j)\Delta\tau \hat{H}_M} e^{-i\tau_j \Delta\tau \hat{H}_C}$
Simulated Adiabatic
$\prod_{l=1}^{P} e^{-i\nu_l \hat{H}_M} e^{-i\gamma_l \hat{H}_C}$
QAOA

Figure 2.3: Conceptual analogy for comparing analog (continuous) adiabatic, simulated (discretised) adiabatic evolution and QAOA (discretised and re-parameterised) as a path through state space [29].

The QAOA algorithm involves repeating the following procedure:

1. Preparation of the highest energy state $|s\rangle$ of $\hat{\mathcal{H}}_{\mathcal{B}}$

2. Applications of the *mixing* $\hat{\mathcal{U}}_{\mathcal{B}}$ and *problem* $\hat{\mathcal{U}}_{\mathcal{C}}$ unitaries to form

$$\left|\underline{\gamma}, \underline{\beta}\right\rangle_p := \prod_{i=0}^{p-1} \hat{\mathcal{U}}_{\mathcal{B}}(\beta_i)\hat{\mathcal{U}}_{\mathcal{C}}(\gamma_i)\left|s\right\rangle \tag{2.43}$$

3. Calculating

$$F_p := \left\langle \underline{\gamma}, \underline{\beta}\right| \hat{\mathcal{H}}_{\mathcal{C}} \left|\underline{\gamma}, \underline{\beta}\right\rangle_p \tag{2.44}$$

and updating $\underline{\gamma} = \gamma_1, \dots, \gamma_p$ and $\underline{\beta} = \beta_1, \dots, \beta_p$ in order to maximise F_p.

Let

$$M_p = \max_{\underline{\gamma}, \underline{\beta}} F_p \tag{2.45}$$

We note that

$$M_{p+1} \geq M_p \tag{2.46}$$

because increasing the depth of the circuit only increases the size of the accessible Hilbert space. By adiabatic considerations [8]:

$$\lim_{p\to\infty} M_p = \max_{\underline{x}\in\{0,1\}^n} C(\underline{x}) \tag{2.47}$$

2.3.5 Choosing Hamiltonians

While it is clear that our *problem* Hamiltonian $\hat{\mathcal{H}}_{\mathcal{C}}$ should represent our objective function (we explore how to do this in 2.4), it is not immediately obvious what form our *mixer* Hamiltonian $\hat{\mathcal{H}}_{\mathcal{B}}$ should take.

Theorem 2.3.4. $[\hat{\mathcal{H}}_A, \hat{\mathcal{H}}_B] = 0 \Rightarrow \hat{\mathcal{H}}_A, \hat{\mathcal{H}}_B$ *share a common eigenbasis.*

Proof. Let $\left|a\right\rangle$ an eigenstate of $\hat{\mathcal{H}}_A$ with eigenvalue a.

$$\begin{aligned}
[\hat{\mathcal{H}}_A, \hat{\mathcal{H}}_B] = 0 &\Rightarrow [\hat{\mathcal{H}}_A, \hat{\mathcal{H}}_B]\left|a\right\rangle = 0 \\
&\Rightarrow (\hat{\mathcal{H}}_A\hat{\mathcal{H}}_B - \hat{\mathcal{H}}_B\hat{\mathcal{H}}_A)\left|a\right\rangle = 0 \\
&\Rightarrow \hat{\mathcal{H}}_A\hat{\mathcal{H}}_B\left|a\right\rangle - a\hat{\mathcal{H}}_B\left|a\right\rangle = 0 \\
&\Rightarrow \hat{\mathcal{H}}_A(\hat{\mathcal{H}}_B\left|a\right\rangle) = a(\hat{\mathcal{H}}_B\left|a\right\rangle)
\end{aligned} \tag{2.48}$$

I.e. $\hat{\mathcal{H}}_B\left|a\right\rangle$ is also an eigenstate of $\hat{\mathcal{H}}_A$ with eigenvalue a. If $\hat{\mathcal{H}}_A$ has a non-degenerate spectrum then $\exists b : \hat{\mathcal{H}}_B\left|a\right\rangle = b\left|a\right\rangle$, and if not then a linear superposition of $\{\left|a_i\right\rangle : \hat{\mathcal{H}}_A\left|a_i\right\rangle = a\left|a_i\right\rangle\}$ can be found that is an eigenstate of $\hat{\mathcal{H}}_B$. □

As such, it is important that the *mixer* Hamiltonian $\hat{\mathcal{H}}_{\mathcal{B}}$ is chosen to not commute with the *problem* Hamiltonian $\hat{\mathcal{H}}_{\mathcal{C}}$. Otherwise, only the global phases of states in the circuit would be affected. Explicitly, if the Hamiltonians commute then

$$\begin{aligned}
\prod_{j=0}^{p-1} \hat{\mathcal{U}}_{\mathcal{B}}(\beta_j)\hat{\mathcal{U}}_{\mathcal{C}}(\gamma_j)\left|s\right\rangle &= \prod_{j=0}^{p-1} e^{-i\beta_j\hat{\mathcal{H}}_{\mathcal{B}}} e^{-i\gamma_j\hat{\mathcal{H}}_{\mathcal{C}}} \sum_\lambda s_\lambda\left|\lambda\right\rangle \\
&= \prod_{j=0}^{p-1} e^{-i\beta_j\lambda_b} e^{-i\gamma_j\lambda_c} \sum_\lambda s_\lambda\left|\lambda\right\rangle \\
&= \underbrace{\prod_{j=0}^{p-1} e^{-i\beta_j\lambda_b} e^{-i\gamma_j\lambda_c}}_{\text{global phase}}\left|s\right\rangle
\end{aligned} \tag{2.49}$$

where we have expanded $|s\rangle = \sum_\lambda s_\lambda |\lambda\rangle$ in the shared eigenbasis: $\hat{\mathcal{H}}_\mathcal{C} |\lambda\rangle = \lambda_c |\lambda\rangle$, $\hat{\mathcal{H}}_\mathcal{B} |\lambda\rangle = \lambda_b |\lambda\rangle$. Global phases would result in no impact to the measurement outcomes and make QAOA inexpressive (unable to explore the Hilbert space).

While not necessary, many QAOA algorithms on n qubits choose

$$\hat{\mathcal{H}}_\mathcal{B} = \sum_{j=1}^{n} X_j \tag{2.50}$$

where X_j is the multi-qubit Pauli-X operator acting on the j^{th} qubit [8].

The corresponding unitary operator

$$\hat{\mathcal{U}}_\mathcal{B}(\beta) = e^{-i\beta\hat{\mathcal{H}}_\mathcal{B}} = e^{-i\beta\sum_{j=1}^{n} X_j} = \prod_{j=1}^{n} e^{-i\beta X_j} = \prod_{j=1}^{n} R_{X_j}(2\beta) \tag{2.51}$$

Where $R_{X_j}(\theta)$ is the operator corresponding to a rotation of the j^{th} qubit by θ about the x-axis on the Bloch sphere and we've used the fact that $[X_j, X_k] = 0$.

$\hat{\mathcal{H}}_\mathcal{B}$'s highest energy state is

$$|+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{\underline{x}\in\{0,1\}^n} |\underline{x}\rangle \tag{2.52}$$

whose form as a uniform superposition of the computational basis states makes it an intuitive choice as a starting point. However, other Hamiltonians can be used, including ones that encode *hard constraints* [30].

2.3.6 Barren Plateaus

Barren Plateaus are a significant issue faced in the training of Variational Quantum Algorithms (VQAs), such as QAOA, or more generally VQEs (2.3.3) [31]. This term describes the phenomenon where the process of parameter optimisation halts at a sub-optimal set of values (a local minimum). This occurs when the training landscape is flat such that no descent direction can be found: a plateau.

Definition 2.3.2. *(Ansatz) An ansatz is a quantum circuit parameterised by variable angles, it is expressed as a parameterised unitary for some $D \in \mathbb{N}^+$*

$$U(\underline{\theta}) := \prod_{j=1}^{D} U_j(\theta_j) W_j \tag{2.53}$$

where $\{W_j\}_{j=1}^N$ is a chosen set of fixed unitaries and $U_j = e^{-i\theta_j V_j}$ is a rotation parameterised by θ_j and generated by Hermitian operator V_j.

Definition 2.3.3. *(VQA cost function) A VQA cost function takes the form*

$$C_{\rho,H} := Tr[HU(\underline{\theta})\rho U(\underline{\theta})^\dagger] \tag{2.54}$$

where ρ is an n-qubit input state, H a Hermitian operator and $U(\underline{\theta})$ takes the form in 2.3.2.

If $\rho = |\psi\rangle\langle\psi|$, a pure state

$$Tr[HU(\underline{\theta})\rho U(\underline{\theta})^\dagger] = \langle\psi|U(\underline{\theta})^\dagger H U(\underline{\theta})|\psi\rangle \tag{2.55}$$

We recognise that this is the form of the QAOA objective function (2.37), and that the QAOA circuit takes the form of an ansatz. As such, we consider the following results derived in the work of Holmes et. al [32].

Theorem 2.3.5. *(Average of gradient) For a generic ansatz of the form 2.3.2, and cost function $C_{\rho,H}$ of the form 2.3.3, the average of $\partial_k C_{\rho,H}$ over all parameters $\underline{\theta}$ vanishes [32]*

$$\mathbb{E}_{\underline{\theta}}[\partial_k C_{\rho,H}] = 0 \tag{2.56}$$

where

$$\partial_k C_{\rho,H} := \frac{\partial C_{\rho,H}(\underline{\theta})}{\partial \theta_k} \tag{2.57}$$

Theorem 2.3.6. *(Deviation of gradient) For $\delta \geq 0$ [32]*

$$P(|\partial_k C_{\rho,H}| \geq \delta) \leq \frac{Var[\partial_k C_{\rho,H}]}{\delta^2} \tag{2.58}$$

where

$$Var[\partial_k C_{\rho,H}] = \mathbb{E}_{\underline{\theta}}[(\partial_k C_{\rho,H})^2] - \mathbb{E}_{\underline{\theta}}[\partial_k C_{\rho,H}]^2 \tag{2.59}$$

This means that if the variance of partial derivatives is small for all θ_k then the probability that the partial derivative is non-zero is also small for all θ_k. Such a situation is susceptible to Barren Plateaus during optimisation. In particular, this occurs in *expressive* circuits consisting of multiple unitaries that allow a large part of the effective Hilbert space to be explored [32]. We consider the effects of Barren Plateaus in QAOA in Section 3.2.

2.4 Representing Functions as Hamiltonians

The notion of *representing* a function with a Hamiltonian was introduced in 2.3.1. In this section, we explore this further, focusing on objective functions of the form

$$\mathcal{C} : \{0,1\}^n \to \mathbb{R} \tag{2.60}$$

The results derived will allow us to encode SAT problems into our QAOA *problem* Hamiltonian.

2.4.1 Boolean Functions

Definition 2.4.1. *(Boolean function) The class of Boolean function on n bits takes the form*

$$\mathcal{B}_n := \{f : \{0,1\}^n \to \{0,1\}\} \tag{2.61}$$

We begin by representing $f(\underline{x}) = 1$ and $g(\underline{x}) = 0$ respectively:

$$\begin{aligned} \hat{\mathcal{H}}_f &= \mathbb{I} \\ \hat{\mathcal{H}}_g &= 0 \end{aligned} \tag{2.62}$$

Where $\mathbb{I}$ is the identity operator and 0 the null operator.

It is clear that

$$\begin{aligned} \hat{\mathcal{H}}_f |\underline{x}\rangle &= 1 \times |\underline{x}\rangle = |\underline{x}\rangle \\ \hat{\mathcal{H}}_g |\underline{x}\rangle &= 0 \times |\underline{x}\rangle = 0 \end{aligned} \tag{2.63}$$

Next, we represent $h_j(\underline{x}) = x_j$ where $\underline{x} = x_0 x_1 \ldots x_{n-1}$, making use of the Pauli-Z operator $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$. We consider the multiple qubit version

$$Z_j := I^{\otimes(j-1)} \otimes Z \otimes I^{\otimes(n-j)} \tag{2.64}$$

whose application gives

$$Z_j |\underline{x}\rangle = (-1)^{x_j} |\underline{x}\rangle = (1 - 2x_j) |\underline{x}\rangle \tag{2.65}$$

Letting $\hat{\mathcal{H}}_{h_j} := \frac{1}{2}(\mathbb{I} - Z_j)\,|\underline{x}\rangle$

$$\hat{\mathcal{H}}_{h_j}\,|\underline{x}\rangle = \frac{1}{2}(1-(1-2x_j))\,|\underline{x}\rangle = x_j\,|\underline{x}\rangle \tag{2.66}$$

It is clear $\hat{\mathcal{H}}_{h_j}$ represents h_j. Combining these together, the following composition rules can be derived.

Theorem 2.4.1. *Let $f, g \in \mathcal{B}_n$ represented by $\hat{\mathcal{H}}_f, \hat{\mathcal{H}}_g$, then [28]:*

- $\hat{\mathcal{H}}_{\neg f} = \mathbb{I} - \hat{\mathcal{H}}_f$
- $\hat{\mathcal{H}}_{f\wedge g} = \hat{\mathcal{H}}_f\hat{\mathcal{H}}_g$
- $\hat{\mathcal{H}}_{f\vee g} = \hat{\mathcal{H}}_f + \hat{\mathcal{H}}_g - \hat{\mathcal{H}}_f\hat{\mathcal{H}}_g$
- $\hat{\mathcal{H}}_{f\oplus g} = \hat{\mathcal{H}}_f + \hat{\mathcal{H}}_g - 2\hat{\mathcal{H}}_f\hat{\mathcal{H}}_g$
- $\hat{\mathcal{H}}_{f\Rightarrow g} = \mathbb{I} - \hat{\mathcal{H}}_f + \hat{\mathcal{H}}_f\hat{\mathcal{H}}_g$
- $\hat{\mathcal{H}}_{af+bg} = a\hat{\mathcal{H}}_f + b\hat{\mathcal{H}}_g$

In cases where it is inefficient or difficult to express a function as a composition before finding its representation, more general results exist in terms of Fourier coefficients (see Appendix A).

$f(\underline{x})$	$\hat{\mathcal{H}}_f$	$f(\underline{x})$	$\hat{\mathcal{H}}_f$
x	$\frac{1}{2}I - \frac{1}{2}Z$	$\neg x$	$\frac{1}{2}I + \frac{1}{2}Z$
$x_1 \oplus x_2$	$\frac{1}{2}I - \frac{1}{2}Z_1Z_2$	$\bigoplus_{j=1}^{k} x_j$	$\frac{1}{2}I - \frac{1}{2}\prod_{i=1}^{k} Z_k$
$x_1 \wedge x_2$	$\frac{1}{4}I - \frac{1}{4}(Z_1 + Z_2 - Z_1Z_2)$	$\bigwedge_{j=1}^{k} x_j$	$\frac{1}{2^k}\prod_{j}^{k}(1 - Z_j)$
$x_1 \vee x_2$	$\frac{3}{4}I - \frac{1}{4}(Z_1 + Z_2 + Z_1Z_2)$	$\bigvee_{j=1}^{k} x_j$	$I - \frac{1}{2^k}\prod_{j}^{k}(1 + Z_j)$
$\overline{x_1x_2}$	$\frac{3}{4}I + \frac{1}{4}(Z_1 + Z_2 - Z_1Z_2)$	$x_1 \Rightarrow x_2$	$\frac{3}{4}I + \frac{1}{4}(Z_1 - Z_2 + Z_1Z_2)$

Table 2.1: Basic composition rules.

2.4.2 Real/Pseudo-Boolean Functions

In many cases, functions of interest take the form of a (weighted) sum of Boolean functions. We will find that it is useful to be able to express these.

Definition 2.4.2. *(Real function) The class of Real functions on n bits take the form*

$$\mathcal{R}_n := \{f : \{0,1\}^n \to \mathbb{R}\} \tag{2.67}$$

Theorem 2.4.2. *Every $g \in \mathcal{R}_n$ can be written (non-uniquely) as the weighted sum of Boolean functions [28]:*

$$g(\underline{x}) = \sum_j w_j f_j(\underline{x}) \tag{2.68}$$

where $f_j \in \mathcal{B}_{m\leq n}$ (i.e. acts on a subset of the bits) and $w_j \in \mathbb{R}$.

Taking $\mathcal{B}_n$ as the elements of a real vector space, they form a basis of $\mathcal{R}_n$ for each n. As such, the terms f_j, w_j above are effectively basis vectors and their corresponding coefficients in the expansion of the given Real function g. A Real function in this representation is termed *Pseudo-Boolean.*

Theorem 2.4.3. *Let $g \in \mathcal{R}_n$ a Pseudo-Boolean function, its representing Hamiltonian takes the form [28]:*

$$\hat{\mathcal{H}}_g = \sum_j w_j \hat{\mathcal{H}}_{f_j} \tag{2.69}$$

where $\hat{\mathcal{H}}_{f_j}$ is the representing Hamiltonian of $f_j \in \mathcal{B}_n$.

We will find that the objective functions of k-SAT and k-NAE-SAT take *Pseudo-Boolean* forms.

Chapter 3

Related Work

3.1 QAOA Success Probabilities

The recent work of Boulebnane & Montanaro [10] explores the *success probability* of *fixed angle, constant depth* QAOA for solving k-SAT. While QAOA is often used for finding approximate solutions, this work examines its application as an exact solver. Namely, the authors consider repeatedly running QAOA until a measurement outcome on the output state corresponds to a satisfying assignment.

This translates into an exact, yet incomplete, algorithm for solving SAT. If no satisfying assignment exists then no measurement outcome will be a solution and the algorithm will infinitely loop. Nevertheless, incomplete solvers are common in classical algorithms (2.2.2) and are either run on necessarily satisfiable instances or with a finite limit on the number of attempts.

In this section, we summarise their findings and introduce relevant definitions.

Definition 3.1.1. *(Random k-SAT generation). Let $k \in \mathbb{N}^+$ and $r > 0$. A random k-SAT $\underline{\sigma} \sim CNF(n, k, r)$ instance is constructed as follows [10]:*

- *Sample $m \sim Poisson(rn)$*
- *Generate $\underline{\sigma} := \bigwedge_{i=0}^{m-1} \sigma_i$, $\underline{\sigma} \sim F_k(n, m)$ as in 2.1.10*

An assignment $\underline{x} \in \{0,1\}^n$ satisfying $\underline{\sigma}$ is denoted as

$$\underline{x} \vdash \underline{\sigma} \tag{3.1}$$

Definition 3.1.2. *(Random k-SAT QAOA). Let $k, n \in \mathbb{N}^+, r > 0$ in $\underline{\sigma} \sim CNF(n, k, r)$*

$$\hat{\mathcal{H}}_{\underline{\sigma}} := \sum_{\underline{x} \in \{0,1\}^n} |\{j \in \{0, \dots, m-1\} : \underline{x} \nvdash \sigma_j\}| \, |\underline{x}\rangle\langle\underline{x}| \tag{3.2}$$

represents the objective function counting the number of unsatisfied clauses in $\underline{\sigma}$. For each $m' \in \{0, \dots, m-1\}$

$$\{\hat{\mathcal{H}}_{\underline{\sigma}} = m'\} := \sum_{\substack{\underline{x} \in \{0,1\}^n \\ |\{j \in \{0,\dots,m-1\} : \underline{x} \nvdash \sigma_j\}| = m'}} |\underline{x}\rangle\langle\underline{x}| \tag{3.3}$$

denotes the orthogonal projector onto the eigenspace of $\hat{\mathcal{H}}_{\underline{\sigma}}$ with eigenvalue m'. For $\underline{\beta}, \underline{\gamma} \in \mathbb{R}^p$

$$\left|\Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma})\right\rangle := \prod_{j=0}^{p-1} e^{-\frac{i\beta_j}{2} \sum_{k=0}^{n-1} X_k} e^{-\frac{i\gamma_j}{2} \hat{\mathcal{H}}_{\underline{\sigma}}} |+\rangle^{\otimes n} \tag{3.4}$$

denotes the state prepared by p-layer QAOA for the problem defined by $\hat{\mathcal{H}}_{\underline{\sigma}}$. Here

$$|+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{\underline{x}\in\{0,1\}^n} |\underline{x}\rangle \tag{3.5}$$

is the equal superposition across all possible bitstrings.

In particular $\{\hat{\mathcal{H}}_{\underline{\sigma}} = 0\}$ denotes the orthogonal projector onto the space of satisfying assignments.

An ideal QAOA algorithm prepares a state entirely within this subspace, resulting in measurement outcomes that are only satisfying assignments. For states that are not entirely within this subspace, we consider the probability that a measurement outcome is a satisfying assignment.

Definition 3.1.3. *(Success probability) Let $k, n \in \mathbb{N}^+, r > 0$ in $\underline{\sigma} \sim CNF(n,k,r)$ and $\underline{\beta}, \underline{\gamma} \in \mathbb{R}^p$*

$$p_{succ}(\underline{\beta}, \underline{\gamma}, \underline{\sigma}) = \left\langle \Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma}) \middle| \{\hat{\mathcal{H}}_{\underline{\sigma}} = 0\} \middle| \Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma}) \right\rangle \tag{3.6}$$

For brevity this is denoted as just $p_{succ}(\underline{\sigma})$. Optimising the parameters within the QAOA procedure corresponds to maximising the success probability.

3.1.1 Analytic Derivation

Boulebnane & Montanaro prove the following result on the expectation of p_{succ} over randomly generated k-SAT instances.

Theorem 3.1.1. *(Average-case success probability of random k-SAT QAOA) Let $q, p \in \mathbb{N}^+$ and $\underline{\beta}, \underline{\gamma} \in \mathbb{R}^p$. For $k = 2^q, n \in \mathbb{N}^+$ and $\underline{\gamma}$ sufficiently small (constant independent of n)*

$$\lim_{n\to\infty} \frac{1}{n} \log \mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})] = F(\underline{z}^*) + (k-1) \sum_{\alpha\subset[2p+1]} \left(\frac{\partial F}{\partial z_\alpha}\right)^k \tag{3.7}$$

$[2p+1] = \{1, \ldots, 2p+1\}$ *and* $\underline{z}^* \in \mathbb{C}^{2^{2p+1}}$ *is the unique fixed point of*

$$\begin{aligned} &\mathbb{C}^{2^{2p+1}} \to \mathbb{C}^{2^{2p+1}} \\ &(z_\alpha)_{\alpha\subset[2p+1]} \mapsto \left(-k(\partial_\alpha F((z_{\alpha'})_{\alpha'\subset[2p+1]}))^{k-1}\right)_{\alpha\subset[2p+1]} \end{aligned} \tag{3.8}$$

wherein

$$\begin{aligned} &F : \mathbb{C}^{2^{2p+1}} \to \mathbb{C} \\ &F((z_\alpha)_{\alpha\subset[2p+1]}) \mapsto \log \sum_{s\in\{0,1\}^{2p+1}} b_s \exp\left\{ \frac{1}{2} \sum_{\substack{\alpha\subset[2p+1] \\ \forall j,j'\in\alpha : s_j = s_{j'}}} (-c_\alpha)^{\frac{1}{k}} z_\alpha \right\} \end{aligned} \tag{3.9}$$

where

$$\begin{aligned} b_s &:= \frac{1}{2}(-1)^{\mathbb{1}[s_0\neq s_p]} \prod_{j\in[p]} \left(\cos\frac{\beta_j}{2}\right)^{\mathbb{1}[s_j=s_{j+1}]+\mathbb{1}[s_{2p-j}=s_{2p-j-1}]} \left(i\sin\frac{\beta_j}{2}\right)^{\mathbb{1}[s_j\neq s_{j+1}]+\mathbb{1}[s_{2p-j}\neq s_{2p-j-1}]} \\ c_s &:= (-1)^{\mathbb{1}[p\in\alpha]} \prod_{j\in\alpha, j<p} \left(e^{-\frac{i\gamma_j}{2}} - 1\right) \prod_{j\in\alpha, j>p} \left(e^{\frac{i\gamma_{2p-j}}{2}} - 1\right) \end{aligned} \tag{3.10}$$

$\mathbb{1}$ *is the indicator function.*

The result in 3.1.1 is dependent on *fixed* $p \in \mathbb{N}^+$ and $\underline{\beta}, \underline{\gamma} \in \mathbb{R}^p$. In other words, a QAOA circuit with predetermined depth and angles that are independent of the actual problem instance $\underline{\sigma}$. While this is advantageous, as it does not require re-training of the parameters, it gives a pessimistic estimate of the performance of QAOA. It also does not optimise for different values of n, adding an additional source of inaccuracy. Finally, calculating the unique fixed point is an expensive computation with a complexity of $\mathcal{O}(2^{2p+1})$.

Nevertheless, this provides an important heuristic to predict the performance of QAOA. In particular, the authors show that the success probability is closely related to the running time of the algorithm. As such, a quantum advantage can be achieved if the running times corresponding to this analytic result are shown to outperform classical solvers. We explore these heuristics in the section below, as well as our own work later in the thesis.

Their proof relies on the use of *generalised multinomial sums* which considers expressions of the form

$$\left(\sum_j x_j \right)^n \tag{3.11}$$

The expected success probability of fixed-angle k-SAT QAOA can be cast as such a sum. Its leading exponential contribution is estimated with the use of a saddle point method that requires $\underline{\gamma}$ is small. However, the bound on $\underline{\gamma}$ is not straightforward, and while the authors prove that it is unique and exists, they do not provide an immediate representation of it.

The derivation of their results is involved and beyond the scope of this thesis.

3.1.2 Empirical Validation

Boulebnane & Montanaro validate the derived result by examining the performance of QAOA on instances near the satisfiability threshold $r \sim r_k$. In each case, the parameters $\underline{\beta}$ and $\underline{\gamma}$ are selected by training on a subset of 100 instances. A larger set of size 10000 is then used to evaluate the empirical success probability.

Definition 3.1.4. *(Empirical success probability) Given the ensemble* $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,r)\}_{i=0}^{v-1}$, *the QAOA empirical success probability is defined as*

$$\hat{p}_{succ} := \frac{1}{v} \sum_{i=0}^{v-1} p_{succ}(\underline{\sigma}_i) \tag{3.12}$$

In this case $v = 10000$. For $k = 8$, $p \in \{1, 2, 4, 8, 30, 60\}$ and $n \in \{12, \dots, 20\}$. They find that the predicted and empirical success probabilities closely agree, as in Figure 3.1. This provides confidence in the derived result.

We note the choice of parameters: the values of n are such that all QAOA states can be calculated exactly and so classically simulated while the choice of k fits the requirement (3.1.1) that $k = 2^q$ for some q. In particular, $k = 16$ would result in instances that are computationally expensive to analyse while $k = 4$ is a regime that classical solvers are very performant in [11].

3.1.3 Benchmarking

Following confidence in the derived analytic result, Boulebnane & Montanaro carry out a series of benchmarking experiments to investigate the potential of QAOA in solving k-SAT problems.

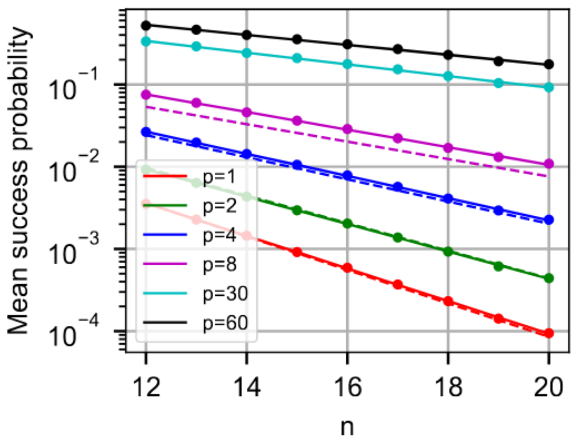


Figure 3.1: Success probability for $k = 8$, varying p. Points are empirical results $\hat{p}_{succ}$ with solid lines fitted. Dashed lines are p_{succ} predicted by theory [10].

3.1.3.1 Running Times

Definition 3.1.5. *(Instance running time) The running time of a random k-SAT QAOA instance $r_{\underline{\sigma}}$ is defined as the number of bitstrings that have to be sampled from the final quantum state $\left|\Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma})\right\rangle$ before one is a satisfying assignment of $\underline{\sigma}$.*

Definition 3.1.6. *(Median running time) The median running time of a random k-SAT QAOA $Med_{\underline{\sigma}\sim CNF(n,k,r)}[r_{\underline{\sigma}}]$ is defined as the median of instance running times $\{r_{\underline{\sigma}} : \underline{\sigma} \sim CNF(n,k,r)\}$.*

The median is used as a benchmark, rather than expected or maximum running time, to mitigate the effect of problem instances that are unsatisfiable. In such cases, it is clear that the instance run time will be infinite, causing the expected run time to be so too. It is also a common method to benchmark classical SAT solvers; allowing for easier comparison [33].

The expected success probability gives a lower bound on the median running time.

Lemma 3.1.1.

$$Med_{\underline{\sigma}\sim CNF(n,k,r)}[r_{\underline{\sigma}}] \geq \frac{1}{2\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})]} \tag{3.13}$$

Proof. Let m_p the median success probability

$$\begin{aligned} \mathbb{E}_{\underline{\sigma}}[p_{succ}(\underline{\sigma})] &= \sum_{\underline{\sigma}} \Pr(\underline{\sigma}) p_{succ}(\underline{\sigma}) \\ &= \sum_{\underline{\sigma}:p_{succ}(\underline{\sigma})<m_p} \Pr(\underline{\sigma}) p_{succ}(\underline{\sigma}) + \sum_{\underline{\sigma}:p_{succ}(\underline{\sigma})\geq m_p} \Pr(\underline{\sigma}) p_{succ}(\underline{\sigma}) \\ &\geq \sum_{\underline{\sigma}:p_{succ}(\underline{\sigma})\geq m_p} \Pr(\underline{\sigma}) p_{succ}(\underline{\sigma}) \\ &\geq \frac{1}{2} m_p \end{aligned} \tag{3.14}$$

It follows

$$\mathbb{E}_{\underline{\sigma}}[p_{succ}(\underline{\sigma})]^{-1} \leq \frac{2}{m_p} \leq 2Med_{\underline{\sigma}}[r_{\underline{\sigma}}] \tag{3.15}$$

where we have used Jensen's inequality for medians [34]. □

The authors assess the relationship

$$Med_{\underline{\sigma}\sim CNF(n,k,r)}[r_{\underline{\sigma}}] \sim [\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})]]^{-1} \tag{3.16}$$

in more detail for $k = 8$, by considering the empirical median running time.

Definition 3.1.7. *(Empirical median running time) Given the ensemble $R = \{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,r)\}_{i=0}^{v-1}$, the QAOA empirical median running time $Med_{\underline{\sigma}\in R}[r_{\underline{\sigma}}]$ is defined as the median of the corresponding running times $\{r_{\underline{\sigma}_i} : \underline{\sigma}_i \in R\}$.*

Again, the ensemble size is $v = 10000$ and $r \sim r_k$. As shown in Figure 3.2, they observe that for small p, the two measures are well aligned, while the slopes differ for larger p. They hypothesise that this is due to the procedure used in selecting the QAOA parameters $\underline{\beta}, \underline{\gamma}$.

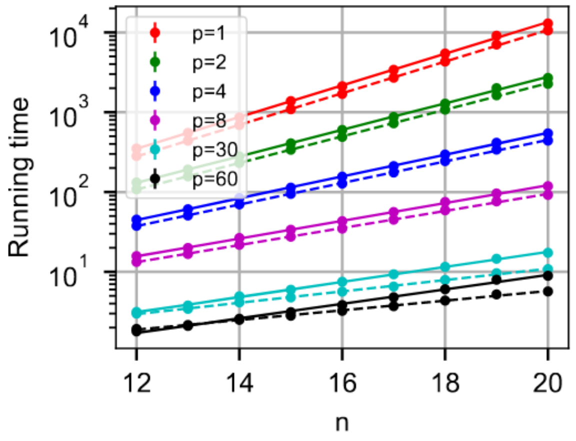


Figure 3.2: Empirical median running time $Med_{\underline{\sigma}\in R}[r_{\underline{\sigma}}]$ (solid line) compared with running time estimated from average success probability $[\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})]]^{-1}$ (dashed line) [10].

3.1.3.2 Classical Benchmarking

The authors compare the median running time of QAOA for $p = 14$ and $p = 60$ to different classical algorithms for $k = 8$. The instance running time for classical solvers is defined as the number of evaluations of the Boolean formula made during solving. As seen in Figure 3.3, QAOA with $p \geq 14$ outperforms the classical solvers.

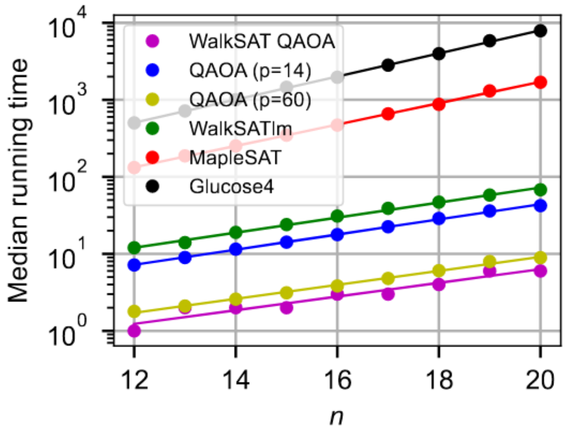


Figure 3.3: Median running times of selected quantum and classical algorithms [10].

3.1.3.3 Scaling Exponents

We are interested in the scaling of QAOA's performance (with respect to the problem size n), particularly in the large n limit. This is considered formally in 3 contexts as follows.

Definition 3.1.8. *(Predicted scaling exponent)*

$$C_{p,k}(\underline{\beta},\underline{\gamma}) = -\lim_{n\to\infty}\frac{1}{n}\log\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})] \tag{3.17}$$

Upon rearrangement:

$$\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})] \sim 2^{-nC_{p,k}(\underline{\beta},\underline{\gamma})} \tag{3.18}$$

Definition 3.1.9. *(Empirical success probability scaling exponent) Given the ensemble* $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,r)\}_{i=0}^{v-1}$, $\hat{C}_{p,k}(\underline{\beta},\underline{\gamma})$ *is such that*

$$\hat{p}_{succ} = \frac{1}{v}\sum_{i=0}^{v-1} p_{succ}(\underline{\sigma}_i) = 2^{-n\hat{C}_{p,k}(\underline{\beta},\underline{\gamma})} \tag{3.19}$$

Definition 3.1.10. *(Empirical median running time scaling exponent) Given the ensemble* $R = \{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,r)\}_{i=0}^{v-1}$, $\tilde{C}_{p,k}(\underline{\beta},\underline{\gamma})$ *is such that*

$$Med_{\underline{\sigma}\in R}[r_{\underline{\sigma}}] = 2^{n\tilde{C}_{p,k}(\underline{\beta},\underline{\gamma})} \tag{3.20}$$

Since

$$\lim_{v\to\infty} \hat{p}_{succ} = \mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})] \tag{3.21}$$

and the results in the previous section show

$$Med_{\underline{\sigma}\sim CNF(n,k,r)}[r_{\underline{\sigma}}] \sim [\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})]]^{-1} \tag{3.22}$$

we expect

$$C_{p,k}(\underline{\beta},\underline{\gamma}) \approx \hat{C}_{p,k}(\underline{\beta},\underline{\gamma}) \approx \tilde{C}_{p,k}(\underline{\beta},\underline{\gamma}) \tag{3.23}$$

To this end, Boulebnane & Montanaro focus on these induced exponents in comparison to `WalkSATlm`, the classical solver found to have the lowest median running time (3.3). The experiments were carried out for $n \in \{12,\dots,20\}$, $k = 8$ and

$$C_{p,k}(\underline{\beta},\underline{\gamma}) \text{ for } p \leq 10 \tag{3.24}$$

$$\hat{C}_{p,k}(\underline{\beta},\underline{\gamma}),\ \tilde{C}_{p,k}(\underline{\beta},\underline{\gamma}) \text{ for } p \leq 60 \tag{3.25}$$

The authors observe (3.4) that $C_{p,k}(\underline{\beta},\underline{\gamma})$ and $\hat{C}_{p,k}(\underline{\beta},\underline{\gamma})$ closely agree. However, $\tilde{C}_{p,k}(\underline{\beta},\underline{\gamma})$ diverges. They also attribute this to the $\underline{\beta}$ and $\underline{\gamma}$ selection procedure. Nonetheless, in all 3 cases, they find an advantage over `WalkSATlm`. In particular, they calculate the scaling of `WalkSATlm` from Figure 3.1.3.2 and find regimes of p for which the scaling exponents of QAOA are smaller.

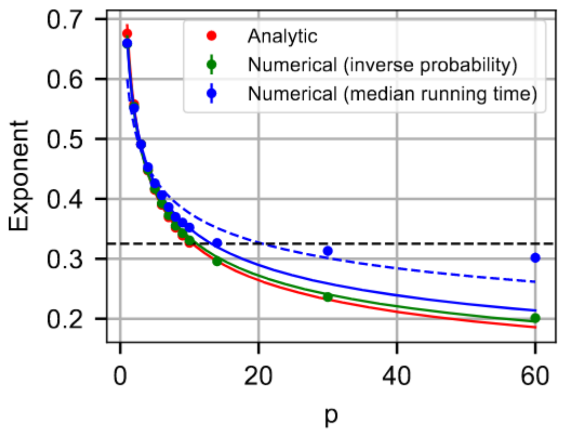


Figure 3.4: Comparison of scaling exponents $C_{p,k}(\underline{\beta},\underline{\gamma})$, $\hat{C}_{p,k}(\underline{\beta},\underline{\gamma})$ and $\tilde{C}_{p,k}(\underline{\beta},\underline{\gamma})$. Dashed black line is empirically estimated median running time of `WalkSATlm`. Dashed blue line is fitting based on all p and solid blue line is fitted on $p \leq 10$ [10].

3.2 Quantum Computational Phase Transitions

The recent work of Zhang et. al [35] explores phase transitions (2.1.3) and Barren Plateaus (2.3.6), within the context of QAOA for k-SAT and 1-in-k-SAT$^+$.

Definition 3.2.1. *(1-in-k-SAT$^+$) Given a propositional formula ϕ in CNF, where each clause has k* ***positive*** *variables (no negations), does there exist an assignment v on ϕ, such that v satisfies ϕ and v assigns* ***exactly one*** *variable true in each clause.*

In particular, the authors focus on instances where $k = 2$ and $k = 3$ as this allows for comparison between the complexity classes (2-SAT, 1-in-2-SAT$^+$ $\in$) P and (3-SAT, 1-in-3-SAT$^+$ $\in$) NP.

3.2.1 Barren Plateaus in Training

Zhang et. al study the relationship between Barren Plateaus and QAOA's ability to solve SAT problems. To do so, they consider different clause densities m/n and evaluate the gradient of the cost function on random choices of circuit parameters. The following standard deviation (SD) is introduced:

$$\left[SD(\partial_1 C(\underline{\gamma}, \underline{\beta}))\right]^{-1} \tag{3.26}$$

where $\partial_1 C(\underline{\gamma}, \underline{\beta})$ is the partial derivative, with respect to parameter γ_1, of $C(\underline{\gamma}, \underline{\beta}) = \langle \underline{\gamma}, \underline{\beta} | \hat{\mathcal{H}}_{\mathcal{C}} | \underline{\gamma}, \underline{\beta} \rangle_p$ (2.37). This allows situations where the variance of the gradient vanishes (and Barren Plateaus are expected) to be easily identified.

The results in Figures 3.5a, 3.5b clearly show that for all problems $\left[SD(\partial_1 C(\underline{\gamma}, \underline{\beta}))\right]^{-1}$ has a clear peak at *some* critical clause density m/n. In general, this isn't the same as the satisfiability threshold, except for 1-in-3-SAT$^+$. This indicates that there is an algorithmic threshold (2.6) for quantum algorithms that is different to that of classical algorithms and to the satisfiability threshold.

3.2.2 Accuracy of QAOA

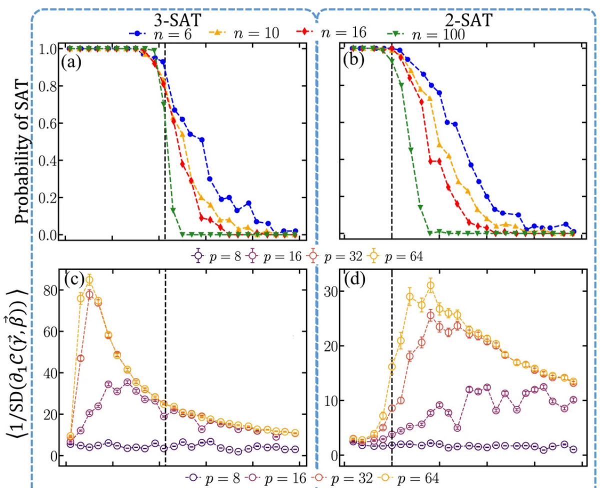
The authors also investigate the accuracy of QAOA in solving SAT problems via the approximation ratio. They compare this to the approximation ratio of the MWIS algorithm [36].

Definition 3.2.2. *(Approximation Ratio) $r \leq 1$ is the ratio between the number of clauses satisfied by a solution and the maximum number of clauses that can be satisfied.*

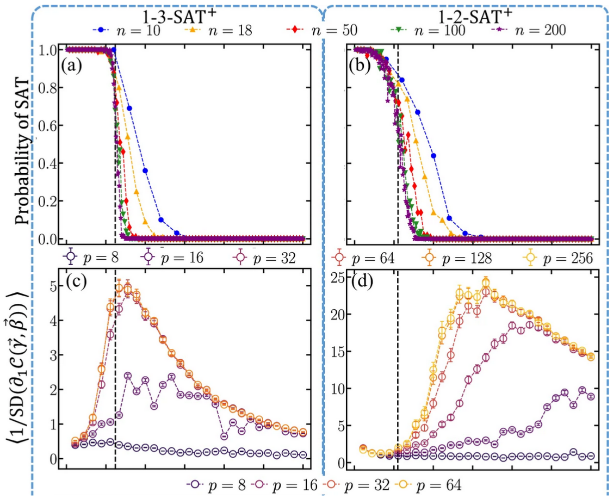
For QAOA, the approximation ratio is found by measuring the output of the circuit 10 times, finding the approximation ratio of each outcome, and keeping the highest amongst the results.

Figures 3.5c, 3.5d show that as p increases, QAOA's approximation ratio does too. On the other hand, as expected, an increase in the clause density corresponds to a decrease in the approximation ratio. However, the decay is rather slow and supports the robustness of QAOA. For MAX-1-3-SAT$^+$, they identify a clear quantum advantage at around $p \approx 16$. For MAX-1-2-SAT$^+$ advantages appear for an even shallower depth of $p \approx 8$ [35].

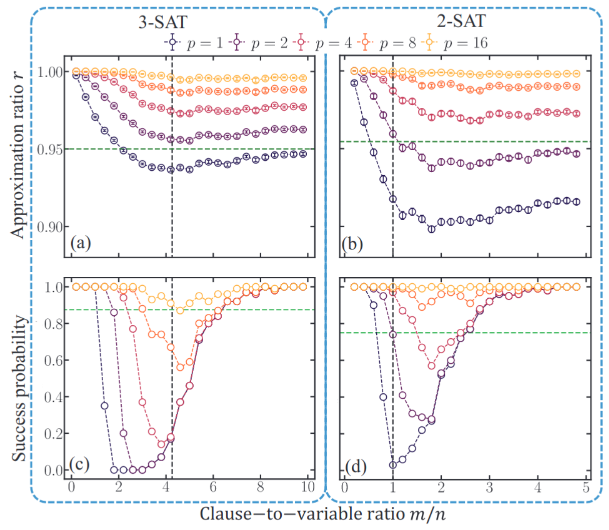
Zhang et. al also recast each optimisation problem as a decision problem by setting a threshold $E_{th} = 0.5$ on the number of satisfied clauses. The instance is considered satisfiable iff the number of clauses the solution satisfies is greater than E_{th}. The probability of successfully classifying the instance for fixed clause densities is analysed. The results of this are seen in Figures 3.5c, 3.5d. As expected, the success probability increases with p. The areas of low success observed are remnants of the classical empirical hardness and are different from the Barren Plateau difficulties investigated in the previous set of results [35].



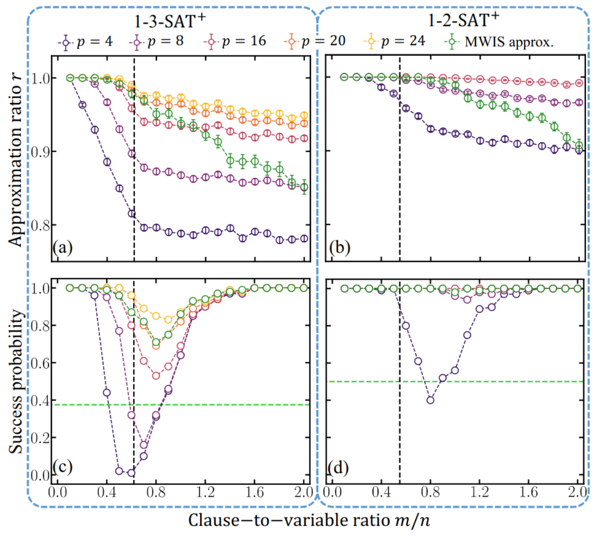
(a) Trainability of k-SAT^{+}: (a, b) Probability problem instance is satisfiable for n variables, (c, d) Mean of $1/SD(\partial_1 C(\underline{\gamma}, \underline{\beta}))$ for $n = 16$. Vertical dashed line represents satisfiability ratio r_k, x-axis is clause density m/n [35].



(b) Trainability of 1-in-k-SAT: (a, b) Probability problem instance is satisfiable for n variables, (c, d) Mean of $1/SD(\partial_1 C(\underline{\gamma}, \underline{\beta}))$ for $n = 16$. Vertical dashed line represents satisfiability ratio r_k, x-axis is clause density m/n [35].



(c) Accuracy of QAOA in k-SAT: (a,b) Approximation ratio r, (c, d) Success probability in determining satisfiability with $n = 10$. Horizontal green line represents success probability of the random guess. Vertical black dashed line represents satisfiability ratio r_k [35].



(d) Accuracy of QAOA in 1-in-k-SAT^{+}: (a,b) Approximation ratio r, (c, d) Success probability in determining satisfiability with $n = 10$. Horizontal green line represents success probability of the random guess. Vertical black dashed line represents satisfiability ratio r_k [35].

Chapter 4

QAOA for k-SAT

The main aim of this section is to independently reproduce the results of Boulebnane & Montanaro [10] and gain confidence in our approach before analysing the k-NAE-SAT problem. To do so, we derive an encoding of k-SAT for QAOA and evaluate its performance on a range of problem instances.

4.1 Implementation

We begin by deriving a formulation that is implementable on any gate-based quantum computer and provide an implementation of the result using `Qiskit` [13]. In addition, for the purposes of efficient large-scale simulation, we consider a *diagonalisation* of our encoding and produce a corresponding performant `PyTorch` [14] procedure.

4.1.1 Problem Hamiltonian

As introduced, QAOA aims to maximise an objective function through its representing Hamiltonian. In the case of k-SAT, the aim is to find a satisfying assignment. This corresponds to maximising the number of satisfied clauses, or equivalently, minimising the number of unsatisfied clauses. We follow conventions in 3.1 and apply the results derived in 2.4 to represent this objective. First, we recall the definition of a random k-SAT instance.

Definition 4.1.1. *(Random k-SAT generation). Let $k \in \mathbb{N}^+$ and $r > 0$. A random k-SAT $\underline{\sigma} \sim CNF(n, k, r)$ instance is constructed as follows:*

- *Sample $m \sim Poisson(rn)$*
- *Generate $\underline{\sigma} \sim F_k(n, m)$ as in 2.1.10, where $\underline{\sigma} := \bigwedge_{i=0}^{m-1} \sigma_i$, $\sigma_i := \bigvee_{j=0}^{k-1} l_{\sigma_{ij}}$*

$l_{\sigma_{ij}}$ is a Boolean literal $l_{\sigma_{ij}} = x_{\sigma_{ij}}$ or $\neg x_{\sigma_{ij}}$ and $\sigma_{ij} \in \{0, \ldots, n-1\}$ is an index into the n variables.

An assignment $\underline{x} \in \{0, 1\}^n$ satisfying $\underline{\sigma}$ is denoted as

$$\underline{x} \vdash \underline{\sigma} \tag{4.1}$$

Minimising the number of unsatisfied clauses corresponds to minimising:

$$\mathcal{C}_{\underline{\sigma}}(\underline{x}) = \sum_{i=0}^{m-1} \mathbb{1}\{\underline{x} \nvdash \sigma_i\} \tag{4.2}$$

where $\underline{x} \in \{0, 1\}^n$ encodes variable assignments and $\mathbb{1}$ is the indicator function.

Example 4.1.1. *Let* $\underline{\sigma} = (x_0 \vee x_1 \vee \neg x_2) \wedge (x_0 \vee x_3 \vee x_4)$*.* $C_{\underline{\sigma}}(01100) = 1$
This is a CNF formula of 5 *variables,* 3 *variables per clause and* 2 *clauses.*
The assignment $v(x_0) = v(x_3) = v(x_4) = \bot$*,* $v(x_1) = v(x_2) = \top$ *corresponds to* $\underline{x} = 01100$*.*
$\mathbb{1}\{\underline{x} \nvdash \underline{\sigma}_0\} = 0$ *because* $v(x_0 \vee x_1 \vee \neg x_2) \equiv v(x_0) \vee v(x_1) \vee v(\neg x_2) \equiv \bot \vee \top \vee \top \equiv \top$*.*
$\mathbb{1}\{\underline{x} \nvdash \underline{\sigma}_1\} = 1$ *because* $v(x_0 \vee x_3 \vee x_4) \equiv v(x_0) \vee v(x_3) \vee v(x_4) \equiv \bot \vee \bot \vee \bot \equiv \bot$*.*

We identify $\mathcal{C}_{\underline{\sigma}}$ as a Pseudo-Boolean function. By Theorem 2.4.3 its representing Hamiltonian takes the form

$$\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}} = \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\mathbb{1}\{\underline{x} \nvdash \sigma_i\}} \equiv \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg \sigma_i} \tag{4.3}$$

where $\hat{\mathcal{H}}_{\mathbb{1}\{\underline{x} \nvdash \sigma_i\}}$ represents each clause and can be written as $\hat{\mathcal{H}}_{\neg \sigma_i}$. This is because each clause can be treated as a Boolean function

$$\sigma_i : \{0,1\}^n \to \{0,1\} \tag{4.4}$$

such that

$$\sigma_i(\underline{x}) = \begin{cases} 1 & \underline{x} \vdash \sigma_i \\ 0 & \underline{x} \nvdash \sigma_i \end{cases} \tag{4.5}$$

By De Morgan's law:

$$\hat{\mathcal{H}}_{\neg \sigma_i} = \hat{\mathcal{H}}_{\neg (\bigvee_{j=0}^{k-1} l_{\sigma_{ij}})} \equiv \hat{\mathcal{H}}_{\bigwedge_{j=0}^{k-1} \neg l_{\sigma_{ij}}} \tag{4.6}$$

Applying composition laws (2.4.1):

$$\hat{\mathcal{H}}_{\bigwedge_{j=0}^{k-1} \neg l_{\sigma_{ij}}} = \prod_{j=0}^{k-1} \hat{\mathcal{H}}_{\neg l_{\sigma_{ij}}} = \prod_{j=0}^{k-1} (\mathbb{I} - \hat{\mathcal{H}}_{l_{\sigma_{ij}}}) \tag{4.7}$$

Recalling (2.1) $\hat{\mathcal{H}}_{f_i} = \frac{1}{2}(\mathbb{I} - Z_i)$ represents $f_i(\underline{x}) = x_i$ and $\hat{\mathcal{H}}_{f_i} = \frac{1}{2}(\mathbb{I} + Z_i)$ represents $f_i(\underline{x}) = \neg x_i$, we combine these to derive that:

$$\hat{\mathcal{H}}_{l_{\sigma_{ij}}} = \frac{1}{2}\mathbb{I} + s_{\sigma_{ij}} \frac{1}{2} Z_{\sigma_{ij}} \tag{4.8}$$

represents $l_{\sigma_{ij}}$, where $s_{\sigma_{ij}} = -1$ for positive literals $l_{\sigma_{ij}} = x_{\sigma_{ij}}$ and $s_{\sigma_{ij}} = 1$ for negative literals $l_{\sigma_{ij}} = \neg x_{\sigma_{ij}}$. Putting this together, we find:

$$\hat{\mathcal{H}}_{\neg \sigma_i} = \prod_{j=0}^{k-1} \left(\mathbb{I} - \hat{\mathcal{H}}_{l_{\sigma_{ij}}}\right) = \prod_{j=0}^{k-1} \left(\mathbb{I} - \frac{1}{2}\mathbb{I} - s_{\sigma_{ij}} \frac{1}{2} Z_{\sigma_{ij}}\right) = \frac{1}{2^k} \prod_{j=0}^{k-1} \left(\mathbb{I} - s_{\sigma_{ij}} Z_{\sigma_{ij}}\right) \tag{4.9}$$

Expanding, and ignoring constants as this is a function being minimised, we arrive at:

$$\hat{\mathcal{H}}_{\neg \sigma_i} = \frac{1}{2^k} \left(- \sum_{a=0}^{k-1} s_{\sigma_{ia}} Z_{\sigma_{ia}} + \sum_{b>a}^{k-1} s_{\sigma_{ia}} s_{\sigma_{ib}} Z_{\sigma_{ia}} Z_{\sigma_{ib}} - \sum_{c>b>a}^{k-1} s_{\sigma_{ia}} s_{\sigma_{ib}} s_{\sigma_{ic}} Z_{\sigma_{ia}} Z_{\sigma_{ib}} Z_{\sigma_{ic}} \dots \right) \tag{4.10}$$

Example 4.1.2. *Let* $\sigma_i = x_a \vee x_b \vee x_c$*, this reduces to:*

$$\begin{aligned} \hat{\mathcal{H}}_{\neg \sigma_i} &= \frac{1}{2^3} \left(\sum_{j=0}^{2} Z_{\sigma_{ij}} + \sum_{j=0}^{2} \sum_{k>j}^{2} Z_{\sigma_{ij}} Z_{\sigma_{ik}} + Z_a Z_b Z_c \right) \\ &= \frac{1}{8} \left(Z_a + Z_b + Z_c + Z_a Z_b + Z_b Z_c + Z_a Z_c + Z_a Z_b Z_c \right) \\ &= \frac{1}{8} \prod_{j=0}^{2} (\mathbb{I} + Z_j) - \frac{1}{8}\mathbb{I} \\ &= \mathbb{I} - \left(\mathbb{I} - \frac{1}{8} \prod_{j=0}^{2} (\mathbb{I} + Z_j) + \frac{1}{8}\mathbb{I} \right) \end{aligned} \tag{4.11}$$

in agreement with 2.1 *up to a constant term.*

Next, we consider the corresponding unitary operator

$$\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}(\gamma) = \exp\left[-i\gamma\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}}\right] = \exp\left[-i\gamma\sum_{j=0}^{m-1}\hat{\mathcal{H}}_{\neg\sigma_j}\right] = \prod_{j=0}^{m-1}\exp\left[-i\gamma\hat{\mathcal{H}}_{\neg\sigma_j}\right] := \prod_{j=0}^{m-1}\hat{\mathcal{U}}_{\neg\sigma_j}(\gamma) \tag{4.12}$$

and find that

$$\begin{aligned}
&\hat{\mathcal{U}}_{\neg\sigma_j}(\gamma)\\
&= \exp\left[-i\gamma\frac{1}{2^k}\left(-\sum_{a=0}^{k-1}\theta_a Z_{\sigma_{ja}} + \sum_{b>a}^{k-1}\theta_{ab}Z_{\sigma_{ja}}Z_{\sigma_{jb}} - \sum_{c>b>a}^{k-1}\theta_{abc}Z_{\sigma_{ja}}Z_{\sigma_{jb}}Z_{\sigma_{jc}}\dots\right)\right]\\
&= \exp\left[i\gamma\frac{1}{2^k}\sum_{a=0}^{k-1}\theta_a Z_{\sigma_{ja}}\right]\exp\left[-i\gamma\frac{1}{2^k}\sum_{b>a}^{k-1}\theta_{ab}Z_{\sigma_{ja}}Z_{\sigma_{jb}}\right]\exp\left[i\gamma\frac{1}{2^k}\sum_{c>b>a}^{k-1}\theta_{abc}Z_{\sigma_{ja}}Z_{\sigma_{jb}}Z_{\sigma_{jc}}\right]\dots\\
&= \prod_{a=0}^{k-1}\exp\left[i\gamma\frac{1}{2^k}\theta_a Z_{\sigma_{ja}}\right]\prod_{b>a}^{k-1}\exp\left[-i\gamma\frac{1}{2^k}\theta_{ab}Z_{\sigma_{ja}}Z_{\sigma_{jb}}\right]\prod_{c>b>a}^{k-1}\exp\left[i\gamma\frac{1}{2^k}\theta_{abc}Z_{\sigma_{ja}}Z_{\sigma_{jb}}Z_{\sigma_{jc}}\right]\dots\\
&= \prod_{a=0}^{k-1}R_{Z_{\sigma_{ja}}}\left[-\frac{\gamma}{2^{k-1}}\theta_a\right]\prod_{b>a}^{k-1}R_{Z_{\sigma_{ja}}Z_{\sigma_{jb}}}\left[\frac{\gamma}{2^{k-1}}\theta_{ab}\right]\prod_{c>b>a}^{k-1}R_{Z_{\sigma_{ja}}Z_{\sigma_{jb}}Z_{\sigma_{jc}}}\left[-\frac{\gamma}{2^{k-1}}\theta_{abc}\right]\dots
\end{aligned} \tag{4.13}$$

where

- $\theta_{ab\dots q} = s_{\sigma_{ja}}s_{\sigma_{jb}}\dots s_{\sigma_{jq}}$
- $R_{Z_1,Z_2,\dots,Z_l}(2\gamma)$ corresponds to the operation $\exp\left[-i\gamma\prod_{j=1}^{l}Z_j\right]$ and is implemented using the decomposition

$$\exp\left[-i\gamma\prod_{j=1}^{l}Z_j\right] = \exp\left[-i\gamma Z^{\otimes l}\right] = \left(\prod_{i=1}^{l-1}\mathrm{CX}_{l-i,l-i+1}\right)R_{Z_l}(2\gamma)\left(\prod_{i=1}^{l-1}\mathrm{CX}_{i,i+1}\right) \tag{4.14}$$

where $\mathrm{CX}_{a,b}$ is the controlled-not gate with control qubit a and target qubit b.

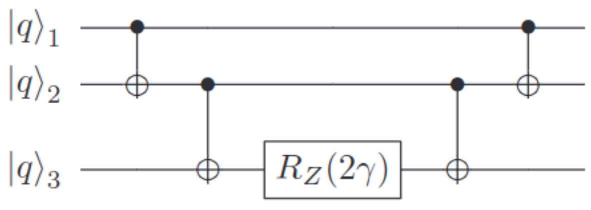


Figure 4.1: Quantum circuit performing the operation $\hat{\mathcal{U}} = \exp\left[-i\gamma Z_1Z_2Z_3\right]$ [28].

4.1.2 QAOA Procedure

The mixer Hamiltonian is as defined in 2.3.5, recall:

$$\hat{\mathcal{H}}_{\mathcal{B}} = \sum_{j=0}^{n-1}X_j \tag{4.15}$$

where X_j is the multi-qubit Pauli-X operator acting on the j^{th} qubit.

The corresponding unitary operator is therefore

$$\hat{\mathcal{U}}_{\mathcal{B}}(\beta) = e^{i\beta\hat{\mathcal{H}}_{\mathcal{B}}} = e^{i\beta\sum\limits_{j=0}^{n-1} X_j} = \prod_{j=0}^{n-1} e^{i\beta X_j} = \prod_{j=0}^{n-1} R_{X_j}(-2\beta) \tag{4.16}$$

Where $R_{X_j}(\theta)$ is the operator corresponding to a rotation of the j^{th} qubit by θ about the x-axis on the Bloch sphere. We note the rotation sign due to the problem being recast as a minimisation.

The output of the QAOA circuit is then given by

$$\left|\Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma})\right\rangle_p := \prod_{i=0}^{p-1} \hat{\mathcal{U}}_{\mathcal{B}}(\beta_i)\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}(\gamma_i)\left|s\right\rangle \tag{4.17}$$

where the initial state $|s\rangle$ is

$$|+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{\underline{x}\in\{0,1\}^n} |\underline{x}\rangle \tag{4.18}$$

$\left|\Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma})\right\rangle_p$ is measured and the corresponding outcome is checked as a satisfiable assignment. This is repeated until a satisfying assignment is found (if one exists). We emphasise that in this context, QAOA is being used an exact algorithm rather than an approximate one (as explored in 2.3.4). The final output of the algorithm is an entirely satisfying assignment and not an approximate solution to the problem.

Extending on the work of Boulebnane & Montanaro (3.1.2), we assess the ability of *fixed angle* QAOA to solve random k-SAT instances, with clause densities near the satisfiability threshold r_k (2.5). Since the threshold is not exactly known, we use the following approximations [6]:

$$\hat{r}_4 = 9.93 \tag{4.19}$$ $$\hat{r}_8 = 176.54 \tag{4.20}$$

The fixed angles are pretrained parameters $\underline{\beta}^*$ and $\underline{\gamma}^*$. For each layer count p, and value of k, they are selected by:

1. Generating $t = 100$ random instances (4.1.1), $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n = 12, k, \hat{r}_k)\}_{i=0}^{t-1}$
2. Initialising $\beta_i = 0.01, \gamma_i = -0.01, \forall i \in \{0, \dots, p-1\}$
3. Optimising $\underline{\beta}, \underline{\gamma}$ over 100 epochs using the `PyTorch` ADAM optimiser to maximise

$$\frac{1}{t}\sum_{i=0}^{t-1} p_{succ}(\underline{\sigma}_i) \tag{4.21}$$

   the success probability over the instances

For clarity, we recall the definition of the success probability (3.6) here:

Definition 4.1.2. *(Success probability) Let* $k, n \in \mathbb{N}^+, r > 0$ *in* $\underline{\sigma} \sim CNF(n, k, r)$ *and* $\underline{\beta}, \underline{\gamma} \in \mathbb{R}^p$

$$p_{succ}(\underline{\beta}, \underline{\gamma}, \underline{\sigma}) = \left\langle\Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma})\right|\{\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}} = 0\}\left|\Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma})\right\rangle_p \tag{4.22}$$

where $\{\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}} = 0\}$ is the projector onto the subspace of states with eigenvalues 0 (satisfying states). For brevity, this is denoted as $p_{succ}(\underline{\sigma})$.

We note that the parameters are selected through training only on $CNF(\mathbf{n = 12}, k, \hat{r}_k)$ instances. In doing so, we assess QAOA's ability to generalise across instances with other variable counts n.

The initial values of $\underline{\beta}, \underline{\gamma}$ were chosen following observations that excessively large angles led to Barren Plateaus (2.3.6). An obvious limitation to this procedure is the small number of instances used to find the optimal values of these parameters. However, Boulebnane & Montanaro justify this by showing the technique provides near-optimal angles (for small p) [10].

To evaluate the circuits, 2500 **satisfiable** random k-SAT instances (3.1.1), $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}_k)\}$ are generated. Importantly, this is done without biasing the procedure (e.g. by selecting variables in such a way that the final formula is guaranteed to be satisfiable). Instead, we randomly generate the instances (as in 3.1.1) then confirm if they are satisfiable using the `Glucose4` solver from `PySAT` [37]. This is an efficient, complete solver (2.2) which allows us to determine with certainty if the instance is satisfiable.

Example 4.1.3. *The propositional formula $\underline{\sigma} \in CNF(3, 3, \frac{8}{3})$, with clauses:*

$$\sigma_0 = \neg x_0 \lor x_1 \lor x_2 \qquad \sigma_3 = \neg x_0 \lor \neg x_1 \lor x_2 \qquad \sigma_6 = x_0 \lor x_1 \lor x_2$$
$$\sigma_1 = x_0 \lor \neg x_1 \lor x_2 \qquad \sigma_4 = \neg x_0 \lor x_1 \lor \neg x_2$$
$$\sigma_2 = x_0 \lor x_1 \lor \neg x_2 \qquad \sigma_5 = x_0 \lor \neg x_1 \lor \neg x_2$$

is satisfied if and only if $\underline{x} = 111$. Training a single layer ($p = 1$) QAOA circuit to maximise $p_{succ}(\underline{\sigma})$ results in the measurement outcomes seen in Figure 4.2. It is clear that 111 is the most sampled outcome with a corresponding empirical success probability of ≈ 0.42.

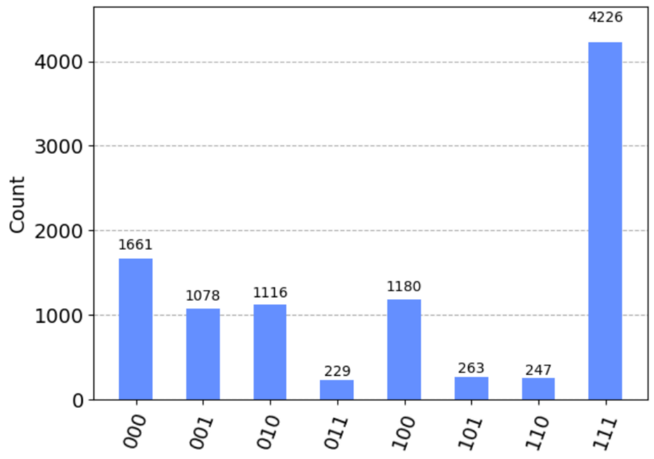


Figure 4.2: 10000 simulated measurements of $|\Psi(\beta, \gamma, \underline{\sigma})\rangle_1$, $\underline{\sigma}$ defined in 4.1.3.

4.1.3 Efficient Classical Simulation

While the unitary operator, $\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}$, derived above is required to implement QAOA on a gate-based quantum computer, it is inefficient for the purposes of simulation. For a system size of $N = 2^n$, states are classically represented as vectors, $z \in \mathbb{C}^N$, and operators as matrices, $\hat{\mathcal{U}} \in \mathbb{C}^{N \times N}$. As such, the application of a gate can be realised as matrix-vector multiplication, requiring $\mathcal{O}(N^2)$ operations. Therefore, we deduce that naively

$$\hat{\mathcal{U}}_{\neg\sigma_i} = \overbrace{\prod_{a=0}^{k-1} \underbrace{R_{Z_{\sigma_{ia}}}(\theta_a)}_{N^2}}^{k} \overbrace{\prod_{b>a}^{k-1} \underbrace{R_{Z_{\sigma_{ia}} Z_{\sigma_{ib}}}(\theta_{ab})}_{3N^2}}^{k-1} \overbrace{\prod_{c>b>a}^{k-1} \underbrace{R_{Z_{\sigma_{ia}} Z_{\sigma_{ib}} Z_{\sigma_{ic}}}(\theta_{abc})}_{5N^2}}^{k-2} \dots \tag{4.23}$$

requires

$$N^2 \sum_{l=0}^{k} (k - l + 1)(2l - 1) = N^2 \frac{1}{6} k(k + 1)(2k + 1) \tag{4.24}$$

operations, where we have used that the decomposition of $R_{Z_1,Z_2,\ldots,Z_l}(\theta)$ (4.14) consists of $2(l-1)+1$ gates. Thus, the application of $\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}$ requires:

$$mN^2\frac{1}{6}k(k+1)(2k+1) = \mathcal{O}\big(mN^2k^3\big) = \mathcal{O}\big(rk^3N^2\log N\big) \tag{4.25}$$

operations since $m = rn$ and $n = \log N$. Near the satisfiability threshold the clause density takes the value $r = r_k \sim 2^k \ln 2$ and so we find the application of $\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}$ requires

$$\mathcal{O}\big(2^k k^3 N^2 \log N\big) \tag{4.26}$$

operations. In the large n limit, N^2 dominates, giving a cost of $\mathcal{O}(N^2)$. However, for the purposes of simulation and, in particular, for cases where $n \sim k$, the pre-factors are computationally relevant. In addition, optimising parameters over the circuit means inefficiencies are aggregated during the automatic differentiation procedure.

Alternatively, inserting a resolution of identity, $\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}}$ can be written as a diagonal operator:

$$\begin{aligned}
\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}} &= \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg\sigma_i} \\
&= \sum_{\underline{x}\in\{0,1\}^n} \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg\sigma_i} \left|\underline{x}\right\rangle \left\langle\underline{x}\right| \\
&= \sum_{\underline{x}\in\{0,1\}^n} \sum_{i=0}^{m-1} \neg\sigma_i(\underline{x}) \left|\underline{x}\right\rangle \left\langle\underline{x}\right| \\
&:= \sum_{\underline{x}\in\{0,1\}^n} h(\underline{x}) \left|\underline{x}\right\rangle \left\langle\underline{x}\right|
\end{aligned} \tag{4.27}$$

In other words,

$$h(\underline{x}) := \sum_{i=0}^{m-1} \neg\sigma_i(\underline{x}) \tag{4.28}$$

is the number of unsatisfied clauses in $\underline{\sigma}$ under the assignment $\underline{x}$.

The corresponding evolution operator $\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}$ is therefore also diagonal and takes the form:

$$\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}(\gamma) = e^{-i\gamma\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}}} = \sum_{\underline{x}\in\{0,1\}^n} e^{-i\gamma h(\underline{x})} \left|\underline{x}\right\rangle \left\langle\underline{x}\right| \tag{4.29}$$

Or, in matrix form

$$\operatorname{diag}\left[h(0), h(1), \ldots, h(N-1)\right] \tag{4.30}$$

where we have written $\underline{x}$ in its decimal representation

$$h(\underline{x}) \equiv h\left(\sum_{l=0}^{n-1} x_l 2^{-l}\right) \tag{4.31}$$

Applying a diagonal matrix only requires $\mathcal{O}(N)$ operations. The calculation of $h(\underline{x})$ requires traversing $m = r\log N \sim 2^k \log N$ clauses and k literals per clause. As such, the application of this diagonalisation only requires

$$\mathcal{O}\big(2^k kN \log N\big) \tag{4.32}$$

operations. Further, the cost of calculating $h(\underline{x})$ can be amortised through preprocessing. As such, we can actually apply $\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}$ in $\mathcal{O}(N)$ operations.

Similarly, the naive application of

$$\hat{\mathcal{U}}_{\mathcal{B}}(\beta) = \overbrace{\prod_{j=0}^{n-1} \underbrace{R_{X_j}(-2\beta)}_{N^2}}^{n} \tag{4.33}$$

requires $\mathcal{O}(N^2 \log N)$ operations, where $n = \log N$. However, writing

$$\hat{\mathcal{U}}_{\mathcal{B}}(\beta) = \prod_{j=0}^{n-1} e^{i\beta X_j} = \prod_{j=0}^{n-1} [\cos(\beta)\mathbb{I} + i\sin(\beta)X_j] \tag{4.34}$$

we realise that $\hat{\mathcal{U}}_{\mathcal{B}}$ can be implemented in $\mathcal{O}(N \log N)$ operations. First,

$$\begin{aligned}[\cos(\beta)\mathbb{I} + i\sin(\beta)X_j] \left|\underline{x}\right\rangle &= \cos(\beta) \underbrace{\mathbb{I} \left|x_0 \dots x_{n-1}\right\rangle}_{0 \text{ operations}} + i\sin(\beta) \underbrace{X_j \left|x_0 \dots x_{n-1}\right\rangle}_{\mathcal{O}(1) \text{ operations}} \\ &= \underbrace{\cos(\beta) \left|x_0 \dots x_{n-1}\right\rangle}_{N \text{ operations}} + \underbrace{i\sin(\beta) \left|x_0 \dots \bar{x}_{j-1} \dots x_{n-1}\right\rangle}_{N \text{ operations}}\end{aligned} \tag{4.35}$$

where $\bar{x}$ denotes the flipped value of x ($\bar{0} = 1, \bar{1} = 0$) and we identify that we can apply

- X_j in $\mathcal{O}(1)$ operations by realising that

$$X_j \left|x_0 \dots x_{n-1}\right\rangle = \left|x_0 \dots \bar{x}_{j-1} \dots x_{n-1}\right\rangle \equiv X_j \left|\underline{x}\right\rangle = \left|2^{n-j} - \underline{x}\right\rangle \tag{4.36}$$

  where we have again written $\underline{x}$ in its decimal representation (4.31). As such, applying the operator corresponds to swapping 2 elements in the corresponding vector $z \in \mathbb{C}^N$.

- $\cos(\beta), \sin(\beta)$ in N operations by a component wise multiplication of the vector $z \in \mathbb{C}^N$.

Therefore

$$\hat{\mathcal{U}}_{\mathcal{B}}(\beta) = \overbrace{\prod_{j=0}^{n-1} \underbrace{[\cos(\beta)\mathbb{I} + i\sin(\beta)X_j]}_{\mathcal{O}(N)}}^{n} \tag{4.37}$$

requires $\mathcal{O}(N \log N)$ operations, where $n = \log N$.

We emphasise that this is only a valid procedure within classical simulations. It is not feasible to directly encode such a diagonal unitary on a gate based quantum computer. Nonetheless, this does not pose an issue for success probability or run time analysis. In particular, the output of both encodings (considered as some $z \in \mathbb{C}^N$) is identical. Therefore, since the procedure is based on sampling from the output state, all calculated metrics, including the induced scaling exponents (3.19), will be unaffected.

4.1.4 Software

We identify that general satisfiability problems, including SAT and its variants, fit into a *clause-literal* framework. The instances consist of clauses that have to be satisfied in some combination, e.g. all of them in k-SAT (since the formula is in CNF). Similarly, the clauses consist of literals that have to be satisfied in some combination, e.g. any of them in k-SAT or at least one, but not all, in k-NAE-SAT. As such, we make use of an object oriented approach to encode our problem instances, wherein the responsibility of confirming satisfiability is delegated to the problem instance.

In particular, all problems derive from a base `Formula` class that contains a set of base `Clause` instances. The `Formula` class implements the `is_satisfied` method, accepting a bitstring and returning `True` or `False` to denote whether the instance is satisfied by the assignment the bitstring represents. We create extensions of this base class to represent the requirements of the formula, such as `CNF` which encodes that the formula is satisfied if and only if all the clauses are satisfied. Similarly, extensions of the `Clause` class: `DisjunctiveClause` and `NAEClause` represent each clauses' constraints.

Not only does this allow us to share logic between classes, but it also means we can construct algorithms that are agnostic to problem instance specifics until run time. In addition, we can generate random problem instances using a common selection procedure that simply instantiates the relevant problem class. We validate the correctness of all this, including our algorithms and problem representations, using the Python `unittest` framework.

The simulations are implemented in `Qiskit` [13], for the generalised unitary, and in `PyTorch` [14], for the diagonalised unitary. In addition, using `Condor` [38] on Imperial College's Department of Computing systems, we are able to parallelise up to 300 simultaneous processes, each with 16 threads.

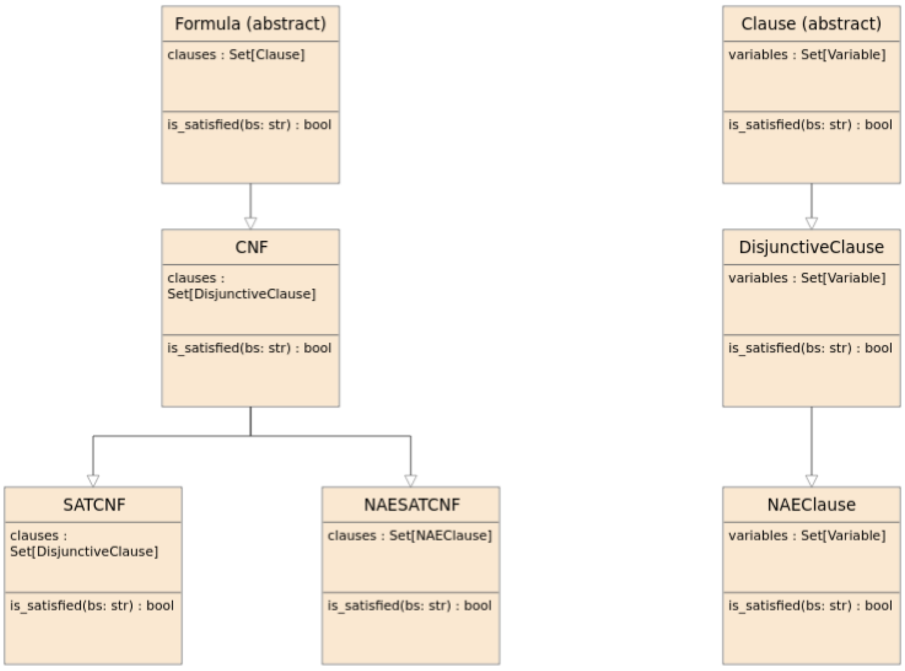

Formula (abstract)
clauses : Set[Clause]
is_satisfied(bs: str) : bool
CNF
clauses : Set[DisjunctiveClause]
is_satisfied(bs: str) : bool
SATCNF
clauses : Set[DisjunctiveClause]
is_satisfied(bs: str) : bool
NAESATCNF
clauses : Set[NAEClause]
is_satisfied(bs: str) : bool
Clause (abstract)
variables : Set[Variable]
is_satisfied(bs: str) : bool
DisjunctiveClause
variables : Set[Variable]
is_satisfied(bs: str) : bool
NAEClause
variables : Set[Variable]
is_satisfied(bs: str) : bool

Figure 4.3: Class diagrams for problem instances.

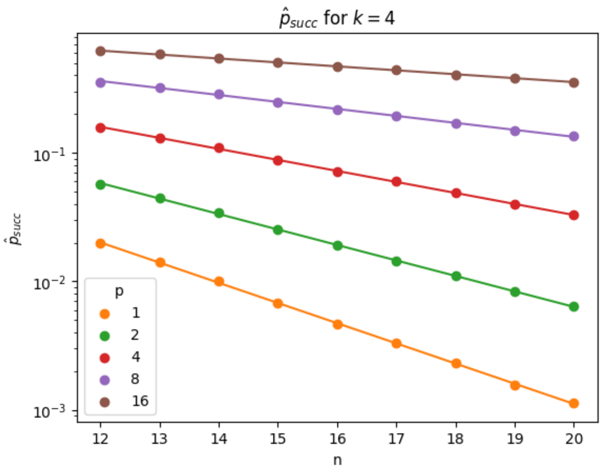
4.2 Evaluation

4.2.1 Success Probabilities

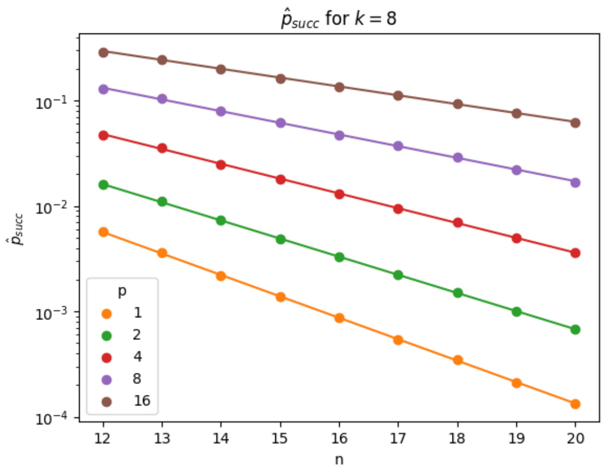
Given parameters $\underline{\beta}^*$ and $\underline{\gamma}^*$, we evaluate the success probability of QAOA over $v = 2500$ **satisfiable** random k-SAT instances, $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}_k)\}_{i=0}^{v-1}$, and calculate

$$\hat{p}_{succ} = \frac{1}{v} \sum_{i=0}^{v-1} p_{succ}(\sigma_i) \tag{4.38}$$

This is done for $p \in \{1, 2, 4, 8, 16\}$ layers, $12 \leq n \leq 20$ and $k \in \{4, 8\}$. The results (Figure 4.4) are found to be in strong agreement with the work of Boulebnane & Montanaro (3.1) and show, as expected, an exponential decay in success probability with instance size. Importantly, the rate of decay decreases as the number of layers p increases.



(a) QAOA average success probabilities across 2500 satisfiable 4-SAT $CNF(n, 4, \hat{r}_4)$ instances (error bars too small to be seen).



(b) QAOA average success probabilities across 2500 satisfiable 8-SAT $CNF(n, 8, \hat{r}_8)$ instances (error bars too small to be seen).

Figure 4.4: QAOA k-SAT success probabilities.

4.2.2 Running Times

Given parameters $\underline{\beta}^*$ and $\underline{\gamma}^*$, we evaluate the running time of QAOA over $v = 2500$ **satisfiable** random instances, $R = \{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}_k)\}_{i=0}^{v-1}$, and calculate the median

$$Med_{\underline{\sigma} \in R}[r_{\underline{\sigma}}] \tag{4.39}$$

recalling (3.1.3.1) that the running time of a random k-SAT QAOA instance $r_{\underline{\sigma}}$ is defined as the number of bitstrings that have to be sampled from the final quantum state $|\Psi(\underline{\beta}, \underline{\gamma}, \underline{\sigma})\rangle$ before one is a satisfying assignment of $\underline{\sigma}$.

This is done for $p \in \{1, 2, 4, 8, 16\}$ layers, $12 \leq n \leq 20$ and $k \in \{4, 8\}$. The results are plotted (Figure 4.5) against the reciprocal of the success probabilities to assess their concordance. Again, the results are found to be in strong agreement with the work of Boulebnane & Montanaro (3.2). In particular, they show both an exponential scaling in median running time with instance size and an alignment with the reciprocal of the success probability. Yet, the slope of the scaling decreases as the number of layers p increases.

We interpret the improved alignment between the two metrics for $k = 8$ as being a result of $k = 4$ instances being small and therefore unchallenging for the QAOA to procedure to solve. This causes the running times to be very similar across instances sizes and results in flat fitting lines.

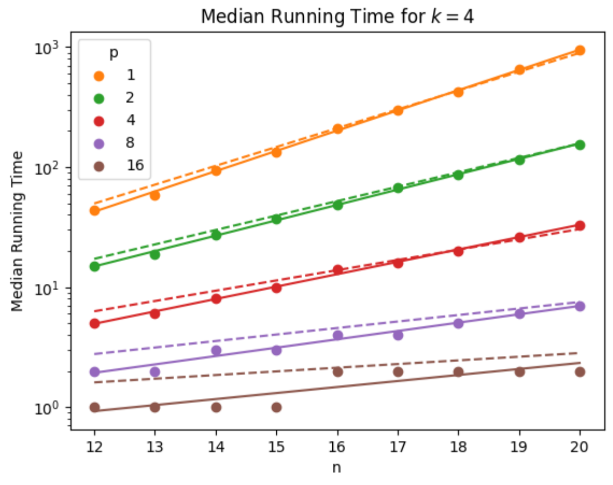
4.2.3 Benchmarking

We compare the median running time for $k = 8, 12 \leq n \leq 19$ of QAOA to that of `WalkSATlm` across 2500 **satisfiable** random k-SAT instances, $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}_k)\}_{i=0}^{v-1}$. In particular, we consider larger values of p to explore the scaling of the run time, where we recall (3.20):

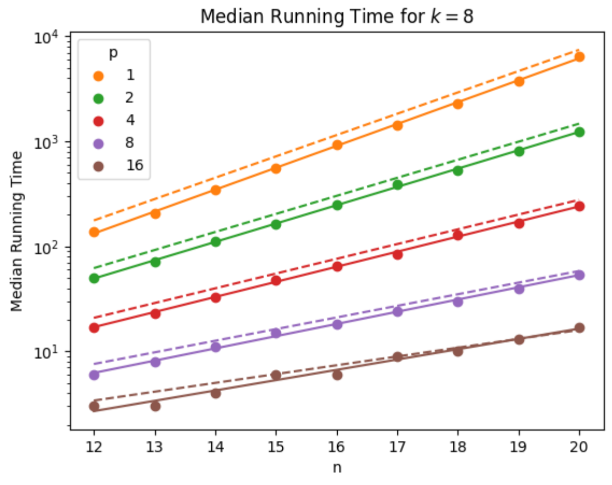
Definition 4.2.1. *(Empirical median running time scaling exponent) Given the ensemble* $R = \{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, r)\}_{i=0}^{v-1}$, $\tilde{C}_{p,k}(\underline{\beta}, \underline{\gamma})$ *is such that*

$$Med_{\underline{\sigma} \in R}[r_{\underline{\sigma}}] = 2^{n\tilde{C}_{p,k}(\underline{\beta}, \underline{\gamma})} \tag{4.40}$$

As introduced in 3, `WalkSATlm` is implemented natively, using suggested hyper-parameters of $p = 0.15, \omega_1 = 6$ and $\omega_2 = 5$ [11]. We define its run time as the number of loop iterations (3) made by



(a) QAOA median running times across 2500 satisfiable 4-SAT $CNF(n,4,\hat{r}_4)$ instances. Dashed lines are the reciprocal of corresponding success probabilities.



(b) QAOA median running times across 2500 satisfiable 8-SAT $CNF(n,8,\hat{r}_8)$ instances. Dashed lines are the reciprocal of corresponding success probabilities.

Figure 4.5: QAOA k-SAT median running times.

the algorithm. Comparing this to QAOA's run time is justified by considering the cost of operations associated with each. For `WalkSATlm`, one iteration involves

$$\mathcal{O}\big(mk^2\big) \tag{4.41}$$

operations to consider the *score* of each of k variables in a clause. Calculating the *score* requires traversing each of the k variables in each of the m clauses.

For QAOA, one sample corresponds to measuring the state

$$\left|\Psi(\underline{\beta},\underline{\gamma},\underline{\sigma})\right\rangle_p := \prod_{i=0}^{p-1} \underbrace{\hat{\mathcal{U}}_{\mathcal{B}}(\beta_i)}_{n \text{ gates}} \underbrace{\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}(\gamma_i)\left|s\right\rangle}_{mk^3 \text{ gates}} \tag{4.42}$$

and so requires

$$\mathcal{O}\big(p(mk^3+n)\big) \tag{4.43}$$

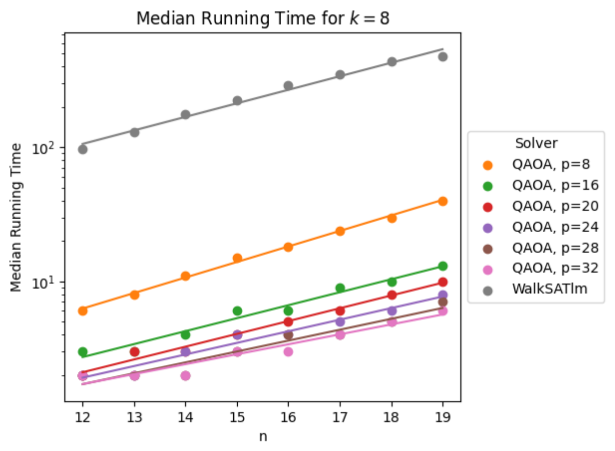
gates (4.25) - where we have absorbed the $\mathcal{O}(n)$ cost of measurement. As $m = rn$, we deduce that for fixed k, p and r, one iteration in both algorithms is an $\mathcal{O}(n)$ operation. Since we are primarily interested in the scaling (3.1.3.3) of the algorithms' running times, which follow an exponential fit $\sim 2^{cn}$, the comparison is valid.

We observe (Figure 4.6a) that the median running time of QAOA outperforms `WalkSATlm` for $k = 8$. Further, we assess the scaling of these running times, $\tilde{C}_{p,k}(\underline{\beta},\underline{\gamma})$, (Figure 4.6b) and find that for both $k = 4$ and $k = 8$, there exists a threshold p above which QAOA's scaling improves on `WalkSATlm`. We fit the coefficients to a power law $\sim ap^b$ and calculate that the coefficients scale as

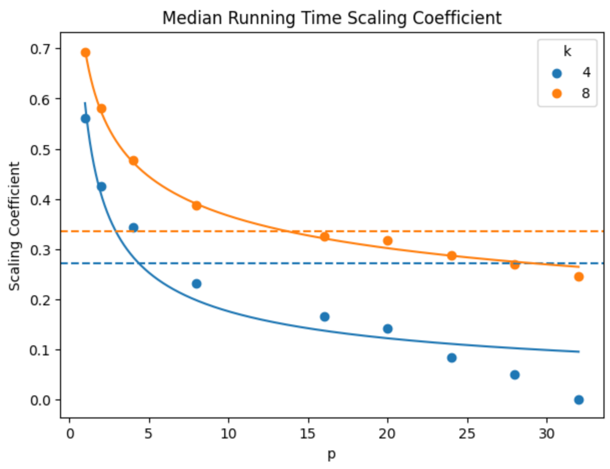
$$\tilde{C}_{p,4}(\underline{\beta},\underline{\gamma}) \sim 0.59p^{-0.53} \tag{4.44}$$

$$\tilde{C}_{p,8}(\underline{\beta},\underline{\gamma}) \sim 0.69p^{-0.28} \tag{4.45}$$

The power law fit for $k = 4$ begins to disagree with the coefficients in the large p limit. Again, this is likely due to the fact that the instances are easy for QAOA to solve regardless of the problem size n. As such, effectively no scaling is induced since the running times are generally ≈ 1. It is clear that this is not the case for $k = 8$ where we find improved agreement due to harder problem instances.



(a) Median running times across 2500 satisfiable 8-SAT $CNF(n, 8, \hat{r}_8)$ instances of QAOA and `WalkSATlm`.



(b) Induced scaling $\tilde{C}_{p,k}(\underline{\beta}, \underline{\gamma})$ of QAOA k-SAT median running times. Dashed line is observed `WalkSATlm` scaling.

Figure 4.6: Benchmarked QAOA k-SAT median running times.

4.2.4 Excessive Scaling

In addition to benchmarking against `WalkSATlm`, we examine the *excessive scaling* [10] of QAOA. In particular, we consider the case where the initial state $|+\rangle^{\otimes n}$ is mostly unchanged by the circuit. We'd expect such behaviour from QAOA when $[\hat{\mathcal{H}}_{\mathcal{B}}, \hat{\mathcal{H}}_{\mathcal{C}}] = 0$ as well as when $||\underline{\beta}||$ or $||\underline{\gamma}|| \ll 1$. In such regimes, QAOA would effectively act as an algorithm that randomly selected bitstrings, with equal probabilities, until a satisfying assignment was found.

We are interested in the scaling of QAOA's success probability in such instances as it allows us to discern its performance from that of the algorithm simply acting as a proxy for random bitstring selection. Recall (3.17):

Definition 4.2.2. *(Predicted scaling exponent)*

$$C_{p,k}(\underline{\beta}, \underline{\gamma}) = -\lim_{n\to\infty} \frac{1}{n} \log \mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})] \tag{4.46}$$

Theorem 4.2.1. *(k-SAT QAOA random assignment scaling exponent)*

$$C_{p,k}(n, \underline{\beta}, \underline{0}) = C_{p,k}(n, \underline{0}, \underline{\gamma}) = 2^{-k}r \tag{4.47}$$

Proof. First, we realise $\forall p$

$$\begin{cases} \left|\Psi(\underline{\beta}, \underline{0}, \underline{\sigma})\right\rangle_p &= \prod_{i=0}^{p-1} \hat{\mathcal{U}}_{\mathcal{B}}(\beta_i) \left|+\right\rangle^{\otimes n} \propto \left|+\right\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{\underline{x}\in\{0,1\}^n} \left|\underline{x}\right\rangle \\ \left|\Psi(\underline{0}, \underline{\gamma}, \underline{\sigma})\right\rangle_p &= \prod_{i=0}^{p-1} \hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}}(\gamma_i) \left|+\right\rangle^{\otimes n} \propto \left|+\right\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{\underline{x}\in\{0,1\}^n} \left|\underline{x}\right\rangle \end{cases} \tag{4.48}$$

since in both cases, the resulting operators all commute and the state is simply effected with a global phase (2.49). As such,

$$p_{succ}(\underline{\sigma}) = \frac{1}{2^n} \sum_{\underline{x},\underline{x}'\in\{0,1\}^n} \langle \underline{x}'| \{\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}} = 0\} |\underline{x}\rangle \tag{4.49}$$

Since $\{\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}} = 0\}$ denotes the orthogonal projector onto the space of satisfying assignments

$$\frac{1}{2^n} \sum_{x,x'\in\{0,1\}} \langle \underline{x}'| \{\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}} = 0\} |\underline{x}\rangle = \frac{1}{2^n} \sum_{\underline{x},\underline{x}'\in\{0,1\}^n} \Pr(\underline{x} \vdash \underline{\sigma})\delta_{\underline{x},\underline{x}'} = \frac{1}{2^n} \sum_{\underline{x}\in\{0,1\}^n} \Pr(\underline{x} \vdash \underline{\sigma}) \tag{4.50}$$

Recall $\underline{\sigma} \sim CNF(n,k,r)$ is a randomly generated problem instance with $m \sim Poisson(rn)$ clauses, such that $\underline{\sigma} := \bigwedge_{i=0}^{m-1} \sigma_i$ and $\sigma_i = \bigvee_{j=0}^{k-1} l_{\sigma_{ij}}$. Therefore,

$$\Pr(\underline{x} \vdash \underline{\sigma}) = \prod_{i=0}^{m-1} \Pr(\underline{x} \vdash \sigma_i) = \prod_{i=0}^{m-1} [1 - \Pr(\underline{x} \nvdash \sigma_i)] = \prod_{i=0}^{m-1} \left[1 - \prod_{j=0}^{k-1} \Pr\left(\underline{x} \nvdash l_{\sigma_{ij}}\right)\right] \tag{4.51}$$

This reduces to

$$\Pr(\underline{x} \vdash \underline{\sigma}) = \left[1 - \left(\frac{1}{2}\right)^k\right]^{m(\underline{\sigma})} \tag{4.52}$$

since all the clauses and literals are chosen independently and satisfying a literal occurs with probability 1/2 as it is negated with equal probability. We write $m(\underline{\sigma})$ to emphasise that m is a property of the random instance $\underline{\sigma}$.

Finally, as $m \sim Poisson(rn)$

$$\begin{aligned}\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}\left[\left(1-2^{-k}\right)^{m(\underline{\sigma})}\right] = \mathbb{E}_{m\sim Poisson(nr)}\left[\left(1-2^{-k}\right)^{m}\right] &= \sum_{m\geq 0} \frac{e^{-rn}(rn)^m}{m!}(1-2^{-k})^m \\ &= e^{-rn}\sum_{m\geq 0}\frac{\left[rn(1-2^{-k})\right]^m}{m!} \\ &= e^{-rn}e^{rn(1-2^{-k})} \\ &= e^{-rn2^{-k}}\end{aligned} \tag{4.53}$$

This means that

$$\lim_{n\to\infty}\frac{1}{n}\log \mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p_{succ}(\underline{\sigma})] = \lim_{n\to\infty}\frac{1}{n}\log e^{-rn2^{-k}} = -2^{-k}r \tag{4.54}$$

and so

$$C_{p,k}(\underline{\beta},\underline{0}) = C_{p,k}(\underline{0},\underline{\gamma}) = 2^{-k}r \tag{4.55}$$

□

We have studied the relationship $C_{p,k}(\underline{\beta},\underline{\gamma}) \approx \hat{C}_{p,k}(\underline{\beta},\underline{\gamma}) \approx \tilde{C}_{p,k}(\underline{\beta},\underline{\gamma})$ and found that they agree well (3.23). As such, we compare $\tilde{C}_{p,k}(\underline{\beta},\underline{\gamma}) = 2^{-k}\hat{r}_k$ to the empirical observations in our experiments. As anticipated, QAOA outperforms the random scaling (Figure 4.7). Interestingly, in the small p regime, the scaling is effectively that of random assignment. This is due to the fact that few-layer circuits are inexpressive, leading to outputs that are effectively unchanged from the initial state.

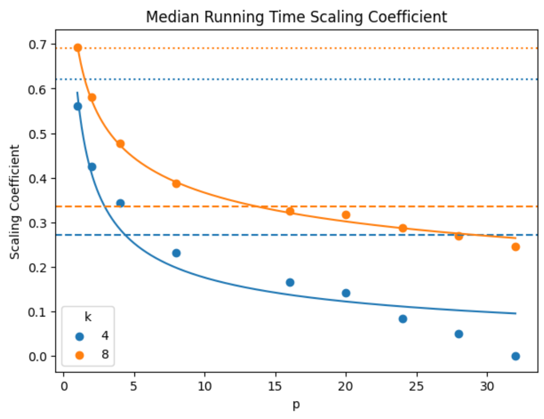


Figure 4.7: Induced scaling of QAOA k-SAT median running times, $\tilde{C}_{p,k}(\underline{\beta},\underline{\gamma})$. Dashed line is observed `WalkSATlm` scaling. Dotted line is random assignment scaling.

Chapter 5

QAOA for k-NAE-SAT

With confidence in our procedure, following the results in the previous section, we derive an encoding of k-NAE-SAT for QAOA and evaluate its performance on a range of problem instances.

5.1 Implementation

Both k-SAT and k-NAE-SAT consider Boolean formulas in Conjunctive Normal Form (CNF). As such, we maintain the formalisms introduced in (4.1.1) and solve problem instances $\underline{\sigma} \sim CNF(n, k, r)$, with m clauses, such that

$$\underline{\sigma} := \bigwedge_{i=0}^{m-1} \sigma_i \tag{5.1}$$

$$\sigma_i = \bigvee_{j=0}^{k-1} l_{\sigma_{ij}} \tag{5.2}$$

Again, $l_{\sigma_{ij}}$ is a Boolean literal $l_{\sigma_{ij}} = x_{\sigma_{ij}}$ or $\neg x_{\sigma_{ij}}$ and $\sigma_{ij} \in \{0, \dots, n-1\}$ is an index into the n variables. Recalling the definition of the k-NAE-SAT problem (2.1.9), the aim is to find a satisfying assignment that sets at least one literal true and at least one literal false in each clause. We denote this

$$\underline{x} \vdash_{\text{NAE}} \underline{\sigma} \tag{5.3}$$

Our objective function to be minimised is therefore

$$\mathcal{C}_{\underline{\sigma}}^{\text{NAE}}(\underline{x}) = \sum_{i=0}^{m-1} \mathbb{1}\{\underline{x} \nvdash_{\text{NAE}} \sigma_i\} \tag{5.4}$$

We note however that

$$\underline{x} \vdash_{\text{NAE}} \sigma_i \equiv \left[\underline{x} \vdash \sigma_i \wedge \underline{x} \nvdash \bigwedge_{j=0}^{k-1} l_{\sigma_{ij}} \right] \tag{5.5}$$

So, by De Morgan's law

$$\underline{x} \nvdash_{\text{NAE}} \sigma_i \equiv \left[\underline{x} \nvdash \sigma_i \vee \underline{x} \vdash \bigwedge_{j=0}^{k-1} l_{\sigma_{ij}} \right] \tag{5.6}$$

In other words, the assignment is penalised if it does not satisfy the clause or if it sets all its literals to true. As such, we can write

$$\mathcal{C}_{\underline{\sigma}}^{\text{NAE}}(\underline{x}) = \sum_{i=0}^{m-1} \mathbb{1}\{\underline{x} \nvdash_{\text{NAE}} \sigma_i\} = \sum_{i=0}^{m-1} \mathbb{1}\{\underline{x} \nvdash \sigma_i \vee \underline{x} \vdash \bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}\} \tag{5.7}$$

We have translated our objective function into one that is directly related to the k-SAT problem. This will allow us to re-use our previously constructed Hamiltonian $\hat{\mathcal{H}}_{\neg\sigma_i}$ (4.10).

5.1.1 Problem Hamiltonian

Again, we identify $\mathcal{C}_{\underline{\sigma}}^{\text{NAE}}$ as a Pseudo-Boolean function (2.4.3). Its representing Hamiltonian takes the form

$$\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{\text{NAE}}} = \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\mathbb{1}\{\underline{x}\nvdash\sigma_i \vee \underline{x}\vdash \bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}\}} \equiv \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg\sigma_i \vee \left(\bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}\right)} \tag{5.8}$$

where we have treated the clauses and literals as Boolean functions.

$$\sigma_i : \{0,1\}^n \to \{0,1\} \tag{5.9}$$

$$l_{ij} : \{0,1\} \to \{0,1\} \tag{5.10}$$

such that

$$\sigma_i(\underline{x}) = \begin{cases} 1 & \underline{x} \vdash \sigma_i \\ 0 & \underline{x} \nvdash \sigma_i \end{cases} \tag{5.11}$$

$$l_{ij}(x) = \begin{cases} 1 & x \vdash l_{ij} \\ 0 & x \nvdash l_{ij} \end{cases} \tag{5.12}$$

Applying composition rules (2.4.1):

$$\hat{\mathcal{H}}_{\neg\sigma_i \vee \left(\bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}\right)} = \hat{\mathcal{H}}_{\neg\sigma_i} + \hat{\mathcal{H}}_{\bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}} - \hat{\mathcal{H}}_{\neg\sigma_i}\hat{\mathcal{H}}_{\bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}} \tag{5.13}$$

The first Hamiltonian, $\hat{\mathcal{H}}_{\neg\sigma_i}$, is as derived for k-SAT (4.10). It is clear that the third Hamiltonian, $\hat{\mathcal{H}}_{\neg\sigma_i}\hat{\mathcal{H}}_{\bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}}$ vanishes, namely $\underline{x}$ cannot both satisfy $\bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}$ while not satisfying $\sigma_i = \bigvee_{j=0}^{k-1} l_{\sigma_{ij}}$. Finally, applying composition rules (2.4.1) and the representation of a literal (4.8), the middle Hamiltonian takes the form

$$\hat{\mathcal{H}}_{\bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}} = \prod_{j=0}^{k-1} \hat{\mathcal{H}}_{l_{\sigma_{ij}}} = \prod_{j=0}^{k-1} \left(\frac{1}{2}\mathbb{I} + s_{\sigma_{ij}}\frac{1}{2}Z_{\sigma_{ij}}\right) = \frac{1}{2^k}\prod_{j=0}^{k-1}\left(\mathbb{I} + s_{\sigma_{ij}}Z_{\sigma_{ij}}\right) \tag{5.14}$$

Expanding, and ignoring constants as this is a function being minimised:

$$\hat{\mathcal{H}}_{\bigwedge_{j=0}^{k-1} l_{\sigma_{ij}}} = \frac{1}{2^k}\left(\sum_{a=0}^{k-1} s_{\sigma_{ia}}Z_{\sigma_{ia}} + \sum_{b>a}^{k-1} s_{\sigma_{ia}}s_{\sigma_{ib}}Z_{\sigma_{ia}}Z_{\sigma_{ib}} + \sum_{c>b>a}^{k-1} s_{\sigma_{ia}}s_{\sigma_{ib}}s_{\sigma_{ic}}Z_{\sigma_{ia}}Z_{\sigma_{ib}}Z_{\sigma_{ic}} \ldots\right) \tag{5.15}$$

The unitary operator corresponding to this is

$$\begin{aligned}
\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}^{\text{NAE}}}(\gamma) = \exp\left[-i\gamma\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{\text{NAE}}}\right] &= \exp\left[-i\gamma\sum_{j=0}^{m-1}\hat{\mathcal{H}}_{\neg\sigma_j} + \hat{\mathcal{H}}_{\bigwedge_{q=0}^{k-1} l_{\sigma_{jq}}}\right] \\
&= \prod_{j=0}^{m-1} \exp\left[-i\gamma\left(\hat{\mathcal{H}}_{\neg\sigma_j} + \hat{\mathcal{H}}_{\bigwedge_{q=0}^{k-1} l_{\sigma_{jq}}}\right)\right] \\
&= \prod_{j=0}^{m-1} \exp\left[-i\gamma\hat{\mathcal{H}}_{\neg\sigma_j}\right]\exp\left[-i\gamma\hat{\mathcal{H}}_{\bigwedge_{q=0}^{k-1} l_{\sigma_{jq}}}\right] \\
&:= \prod_{j=0}^{m-1} \hat{\mathcal{U}}_{\neg\sigma_j}(\gamma)\hat{\mathcal{U}}_{\bigwedge_{q=0}^{k-1} l_{\sigma_{jq}}}(\gamma)
\end{aligned} \tag{5.16}$$

Where we have used the fact that both Hamiltonians only consist of $\mathbb{1}$ and Pauli Z operators, meaning they commute. $\hat{\mathcal{U}}_{\neg\sigma_j}(\gamma)$ is as before, while

$$
\begin{aligned}
&\hat{\mathcal{U}}_{\bigwedge_{q=0}^{k-1} l_{\sigma_{jq}}}(\gamma) \\
&= \exp\left[-i\gamma\frac{1}{2^k}\left(\sum_{a=0}^{k-1}\theta_a Z_{\sigma_{ja}} + \sum_{b>a}^{k-1}\theta_{ab}Z_{\sigma_{ja}}Z_{\sigma_{jb}} + \sum_{c>b>a}^{k-1}\theta_{abc}Z_{\sigma_{ja}}Z_{\sigma_{jb}}Z_{\sigma_{jc}}\dots\right)\right] \\
&= \exp\left[-i\gamma\frac{1}{2^k}\sum_{a=0}^{k-1}\theta_a Z_{\sigma_{ja}}\right]\exp\left[-i\gamma\frac{1}{2^k}\sum_{b>a}^{k-1}\theta_{ab}Z_{\sigma_{ja}}Z_{\sigma_{jb}}\right]\exp\left[-i\gamma\frac{1}{2^k}\sum_{c>b>a}^{k-1}\theta_{abc}Z_{\sigma_{ja}}Z_{\sigma_{jb}}Z_{\sigma_{jc}}\right]\dots \\
&= \prod_{a=0}^{k-1}\exp\left[-i\gamma\frac{1}{2^k}\theta_a Z_{\sigma_{ja}}\right]\prod_{b>a}^{k-1}\exp\left[-i\gamma\frac{1}{2^k}\theta_{ab}Z_{\sigma_{ja}}Z_{\sigma_{jb}}\right]\prod_{c>b>a}^{k-1}\exp\left[-i\gamma\frac{1}{2^k}\theta_{abc}Z_{\sigma_{ja}}Z_{\sigma_{jb}}Z_{\sigma_{jc}}\right]\dots \\
&= \prod_{a=0}^{k-1}R_{Z_{\sigma_{ja}}}\left[\frac{\gamma}{2^{k-1}}\theta_a\right]\prod_{b>a}^{k-1}R_{Z_{\sigma_{ja}}Z_{\sigma_{jb}}}\left[\frac{\gamma}{2^{k-1}}\theta_{ab}\right]\prod_{c>b>a}^{k-1}R_{Z_{\sigma_{ja}}Z_{\sigma_{jb}}Z_{\sigma_{jc}}}\left[\frac{\gamma}{2^{k-1}}\theta_{abc}\right]\dots
\end{aligned}
\tag{5.17}
$$

as in 4.13:

- $\theta_{ab\dots q} = s_{\sigma_{ja}}s_{\sigma_{jb}}\dots s_{\sigma_{jq}}$
- $R_{Z_1,Z_2,\dots,Z_l}(2\gamma)$ corresponds to the operation $\exp\left[-i\gamma\prod_{j=1}^{l}Z_j\right]$ and is implemented using the decomposition

  $$\exp\left[-i\gamma\prod_{j=1}^{l}Z_j\right] = \exp\left[-i\gamma Z^{\otimes l}\right] = \left(\prod_{i=1}^{l-1}\mathrm{CX}_{l-i,l-i+1}\right)R_{Z_l}(2\gamma)\left(\prod_{i=1}^{l-1}\mathrm{CX}_{i,i+1}\right) \tag{5.18}$$

  where $\mathrm{CX}_{a,b}$ is the controlled-not gate with control qubit a and target qubit b.

5.1.2 QAOA Procedure

This proceeds as in the case of k-SAT (4.1.2), where the mixer Hamiltonian is defined:

$$\hat{\mathcal{H}}_{\mathcal{B}} = \sum_{j=0}^{n-1}X_j \tag{5.19}$$

where X_j is the multi-qubit Pauli-X operator acting on the j^{th} qubit.

The corresponding unitary operator is therefore

$$\hat{\mathcal{U}}_{\mathcal{B}}(\beta) = e^{i\beta\hat{\mathcal{H}}_{\mathcal{B}}} = e^{i\beta\sum_{j=0}^{n-1}X_j} = \prod_{j=0}^{n-1}e^{i\beta X_j} = \prod_{j=0}^{n-1}R_{X_j}(-2\beta) \tag{5.20}$$

Where $R_{X_j}(\theta)$ is the operator corresponding to a rotation of the j^{th} qubit by θ about the x-axis on the Bloch sphere. We note the rotation sign due to the problem being recast as a minimisation.

The output of the QAOA circuit is then given by

$$\left|\Psi^{\mathrm{NAE}}(\underline{\beta},\underline{\gamma},\underline{\sigma})\right\rangle_p := \prod_{i=0}^{p-1}\hat{\mathcal{U}}_{\mathcal{B}}(\beta_i)\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}^{\mathrm{NAE}}}(\gamma_i)\left|s\right\rangle \tag{5.21}$$

where the initial state $|s\rangle$ is

$$|+\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}}\sum_{\underline{x}\in\{0,1\}^n}|\underline{x}\rangle \tag{5.22}$$

$\left|\Psi^{\mathrm{NAE}}(\underline{\beta},\underline{\gamma},\underline{\sigma})\right\rangle_p$ is measured and the corresponding outcome is checked as a satisfiable assignment. This is repeated until a satisfying assignment is found (if one exists). Again, QAOA is being used an exact algorithm rather than an approximate one (as explored in 2.3.4). The final output of the algorithm is an entirely satisfying assignment and not an approximate solution to the problem.

We evaluate the effectiveness of this procedure for *fixed angle* QAOA on instances with clause densities near the satisfiability threshold r_k (2.5). Asymptotic values of the threshold are known to be

$$r_k^{\mathrm{NAE}} = \left(2^{k-1} - \frac{1}{2} - \frac{1}{4\log 2}\right)\log 2 + o_k(1) \tag{5.23}$$

where $o_k(1) \to 0$ in the $k \to \infty$ limit [39]. As such, we approximate

$$\hat{r}_k^{\mathrm{NAE}} = \left(2^{k-1} - \frac{1}{2} - \frac{1}{4\log 2}\right)\log 2 \tag{5.24}$$

Next, we extend the definition of the *success probability* introduced for k-SAT.

Definition 5.1.1. *(k-NAE-SAT success probability) Let $k, n \in \mathbb{N}^+, r > 0$ in $\underline{\sigma} \sim CNF(n,k,r)$ and $\underline{\beta},\underline{\gamma} \in \mathbb{R}^p$, the success probability of a random k-NAE-SAT QAOA instance is defined as*

$$p_{succ}^{NAE}(\underline{\beta},\underline{\gamma},\underline{\sigma}) = \left\langle \Psi^{NAE}(\underline{\beta},\underline{\gamma},\underline{\sigma})\right|\{\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{NAE}} = 0\}\left|\Psi^{NAE}(\underline{\beta},\underline{\gamma},\underline{\sigma})\right\rangle_p \tag{5.25}$$

where $\{\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{\mathrm{NAE}}} = 0\}$ is the projector onto the subspace of states with eigenvalues 0 (satisfying states). For brevity, this is denoted as $p_{succ}^{\mathrm{NAE}}(\underline{\sigma})$.

The fixed angles are pretrained parameters $\underline{\beta}^*$ and $\underline{\gamma}^*$. For each layer count p, and value of k, they are selected by:

1. Generating $t = 100$ random instances (4.1.1), $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n = 12, k, \hat{r}_k^{\mathrm{NAE}})\}_{i=0}^{t-1}$

2. Initialising $\beta_i = 0.01, \gamma_i = -0.01, \forall i \in \{0,\ldots,p-1\}$

3. Optimising $\underline{\beta},\underline{\gamma}$ for 100 epochs using the `PyTorch` ADAM optimiser to maximise the success probability over the instances

$$\frac{1}{t}\sum_{i=0}^{t-1} p_{succ}^{\mathrm{NAE}}(\underline{\sigma}_i) \tag{5.26}$$

We emphasise that the parameters are selected through training only on $CNF(\mathbf{n=12}, k, \hat{r}_k^{\mathrm{NAE}})$ instances. In doing so, we assess QAOA's ability to generalise across instances with other variable counts n.

To evaluate the circuits, 2500 **satisfiable** random k-NAE-SAT instances (3.1.1), $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,\hat{r}_k^{\mathrm{NAE}})\}$ are generated. Importantly, this is done without biasing the procedure (e.g. by selecting variables in such a way that the final formula is guaranteed to be satisfiable). Instead, we randomly generate the instances then confirm if they are satisfiable using the `Glucose4` solver from `PySAT` [37]. This is an efficient, complete k-SAT solver (2.2). As such, to use `Glucose4`, we make use of the results in the following section to recast k-NAE-SAT in terms of k-SAT and determine with certainty if the instance is satisfiable.

5.1.3 Efficient Classical Simulation

It is clear that for a system size of $N = 2^n$, the unitary operator derived above has the same complexity as that considered previously: $\mathcal{O}\big(2^k k^3 N^2 \log N\big)$ (4.13). In what follows, we consider an alternative formulation that allows for a more efficient classical simulation. First, we establish a second representation of NAE-SAT.

Lemma 5.1.1. *(Satisfying a k-NAE-SAT clause) For* $\sigma_i = \bigvee_{j=0}^{k-1} l_{\sigma_{ij}}$, $\underline{x} \vdash_{NAE} \sigma_i$ *iff* $\underline{x} \vdash \sigma_i$ ***and*** $\underline{\bar{x}} \vdash \sigma_i$*, where* $\underline{\bar{x}}$ *is the flipped assignment:* $\bar{0} = 1, \bar{1} = 0$.

Proof. "$\Rightarrow$" Suppose $\underline{x} \vdash_{\text{NAE}} \sigma_i$

$$\begin{aligned} &\Rightarrow \exists l_t, l_f \in \sigma_i : \underline{x} \vdash l_t, \underline{x} \nvdash l_f \\ &\Rightarrow \underline{x} \vdash \sigma_i \end{aligned} \tag{5.27}$$

But $\underline{x} \nvdash l_f \Rightarrow \underline{\bar{x}} \vdash l_f \Rightarrow \underline{\bar{x}} \vdash \sigma_i$.

"$\Leftarrow$" Suppose $\underline{x} \vdash \sigma_i$ and $\underline{\bar{x}} \vdash \sigma_i$

$$\begin{aligned} &\Rightarrow \exists l_t \neq l_f \in \sigma_i : \underline{x} \vdash l_t, \underline{\bar{x}} \vdash l_f \\ &\Rightarrow \exists l_t \neq l_f \in \sigma_i : \underline{x} \vdash l_t, \underline{x} \nvdash l_f \\ &\Rightarrow \underline{x} \vdash_{\text{NAE}} \sigma_i \end{aligned} \tag{5.28}$$

□

Consequently

$$\begin{aligned} \underline{x} \nvdash_{\text{NAE}} \sigma_i &\iff \neg(\underline{x} \vdash \sigma_i \wedge \underline{\bar{x}} \vdash \sigma_i) \\ &\iff \underline{x} \nvdash \sigma_i \vee \underline{\bar{x}} \nvdash \sigma_i \end{aligned} \tag{5.29}$$

by De Morgan's law. As such, recalling the initial form of our objective function, we derive

$$\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}(\underline{x}) = \sum_{i=0}^{m-1} \mathbb{1}\{\underline{x} \nvdash_{\text{NAE}} \sigma_i\} = \sum_{i=0}^{m-1} \mathbb{1}\{\underline{x} \nvdash \sigma_i \vee \underline{\bar{x}} \nvdash \sigma_i\} \tag{5.30}$$

The representing Hamiltonian $\hat{\mathcal{H}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}}$ can therefore be written as

$$\hat{\mathcal{H}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}} = \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\mathbb{1}\{\underline{x} \nvdash \sigma_i \vee \underline{\bar{x}} \nvdash \sigma_i\}} \equiv \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg\sigma_i(\underline{x}) \vee \neg\sigma_i(\underline{\bar{x}})} \tag{5.31}$$

Applying the composition rules (2.4.1), we deduce

$$\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x}) \vee \neg\sigma_i(\underline{\bar{x}})} = \hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})} + \hat{\mathcal{H}}_{\neg\sigma_i(\underline{\bar{x}})} - \hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})}\hat{\mathcal{H}}_{\neg\sigma_i(\underline{\bar{x}})} \tag{5.32}$$

Lemma 5.1.2.

$$\hat{\mathcal{H}}_{\neg\sigma_i(\underline{\bar{x}})} = P\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})}P \tag{5.33}$$

where

$$P = \prod_{i=0}^{n-1} X_i \tag{5.34}$$

Proof.

$$P\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})}P\,|\underline{x}\rangle = P\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})}\,|\underline{\bar{x}}\rangle = \neg\sigma_i(\underline{\bar{x}})P\,|\underline{\bar{x}}\rangle = \neg\sigma_i(\underline{\bar{x}})\,|\underline{x}\rangle = \hat{\mathcal{H}}_{\neg\sigma_i(\underline{\bar{x}})}\,|\underline{x}\rangle \tag{5.35}$$

□

Lemma 5.1.3.

$$\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})}\hat{\mathcal{H}}_{\neg\sigma_i(\underline{\bar{x}})} \equiv 0 \tag{5.36}$$

Proof.

$$\begin{aligned}\neg\sigma_i(\underline{x}) = 1 &\Rightarrow \underline{x} \nvdash \sigma_i \\ &\Rightarrow \forall l \in \sigma_i, \underline{x} \nvdash l \\ &\Rightarrow \forall l \in \sigma_i, \bar{\underline{x}} \vdash l \\ &\Rightarrow \bar{\underline{x}} \vdash \sigma_i \\ &\Rightarrow \neg\sigma_i(\bar{\underline{x}}) = 0\end{aligned} \tag{5.37}$$

$$\begin{aligned}\neg\sigma_i(\bar{\underline{x}}) = 1 &\Rightarrow \bar{\underline{x}} \nvdash \sigma_i \\ &\Rightarrow \forall l \in \sigma_i, \bar{\underline{x}} \nvdash l \\ &\Rightarrow \forall l \in \sigma_i, \underline{x} \vdash l \\ &\Rightarrow \underline{x} \vdash \sigma_i \\ &\Rightarrow \neg\sigma_i(\underline{x}) = 0\end{aligned} \tag{5.38}$$

The result immediately follows:

$$\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})}\hat{\mathcal{H}}_{\neg\sigma_i(\bar{\underline{x}})} |\underline{x}\rangle = \neg\sigma_i(\underline{x})\neg\sigma_i(\bar{\underline{x}}) |\underline{x}\rangle = 0 \tag{5.39}$$

□

As such, the Hamiltonian takes the form

$$\hat{\mathcal{H}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}} = \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})\vee\neg\sigma_i(\bar{\underline{x}})} = \sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})} + P\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})}P \tag{5.40}$$

where

$$P = \prod_{i=0}^{n-1} X_i \tag{5.41}$$

and $\hat{\mathcal{H}}_{\neg\sigma_i}$ is as in (4.10). This diagonalises:

$$\begin{aligned}\hat{\mathcal{H}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}} &= \sum_{\underline{x}\in\{0,1\}^n} \left[\sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})} + P\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})}P\right] |\underline{x}\rangle\langle\underline{x}| \\ &= \sum_{\underline{x}\in\{0,1\}^n} \left[\sum_{i=0}^{m-1} \hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})} |\underline{x}\rangle\langle\underline{x}| + P\hat{\mathcal{H}}_{\neg\sigma_i(\underline{x})} |\bar{\underline{x}}\rangle\langle\underline{x}|\right] \\ &= \sum_{\underline{x}\in\{0,1\}^n} \left[\sum_{i=0}^{m-1} \neg\sigma_i(\underline{x})\ |\underline{x}\rangle\langle\underline{x}| + P\neg\sigma_i(\bar{\underline{x}}) |\bar{\underline{x}}\rangle\langle\underline{x}|\right] \\ &= \sum_{\underline{x}\in\{0,1\}^n} \left[\sum_{i=0}^{m-1} \neg\sigma_i(\underline{x}) + \neg\sigma_i(\bar{\underline{x}})\right] |\underline{x}\rangle\langle\underline{x}| \\ &= \sum_{\underline{x}\in\{0,1\}^n} [h(\underline{x}) + h(\bar{\underline{x}})] |\underline{x}\rangle\langle\underline{x}| \\ &:= \sum_{\underline{x}\in\{0,1\}^n} h^{NAE}(\underline{x}) |\underline{x}\rangle\langle\underline{x}|\end{aligned} \tag{5.42}$$

Where $h(\underline{x})$, as introduced previously (4.27), corresponds to the number of unsatisfied clauses in $\underline{\sigma}$ under $\underline{x}$, and by extension we can interpret

$$h^{\text{NAE}}(\underline{x}) := \sum_{i=0}^{m-1} \neg\sigma_i(\underline{x}) + \neg\sigma_i(\bar{\underline{x}}) \tag{5.43}$$

as the number of unsatisfied clauses in $\underline{\sigma}$, under the assignment $\underline{x}$, in the NAE-SAT formulation.

The corresponding evolution operator $\hat{\mathcal{U}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}}$ is therefore also diagonal and takes the form:

$$\hat{\mathcal{U}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}}(\gamma) = e^{-i\gamma\hat{\mathcal{H}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}}} = \sum_{\underline{x}\in\{0,1\}^n} e^{-i\gamma h^{\text{NAE}}(\underline{x})} |\underline{x}\rangle\langle\underline{x}| \tag{5.44}$$

Since $h^{\text{NAE}}(\underline{x}) = h(\underline{x}) + h(\underline{\bar{x}})$, it can be calculated with the same order of cost as before (4.27). Therefore, this application of $\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}^{\text{NAE}}}$ only requires

$$\mathcal{O}\big(2^k kN \log N\big) \tag{5.45}$$

operations. The calculations of $h^{\text{NAE}}(\underline{x})$ can be pre-processed, amortising the cost of $\hat{\mathcal{U}}_{\mathcal{C}_{\underline{\sigma}}^{\text{NAE}}}$ down to $\mathcal{O}(N)$.

Remark 5.1.1. *We note that while the symmetry of h^{NAE}*

$$h^{NAE}(\underline{x}) = h(\underline{x}) + h(\underline{\bar{x}}) = h(\underline{\bar{\bar{x}}}) + h(\underline{\bar{x}}) = h^{NAE}(\underline{\bar{x}}) \tag{5.46}$$

might lead to the problematic presumption that $[\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{NAE}}, \hat{\mathcal{H}}_{\mathcal{B}}] = 0$ (2.3.4), this is not the case. In particular, for arbitrary $|\underline{x}\rangle = |x_0 x_1 \ldots x_{n-1}\rangle$, let $X_i |\underline{x}\rangle = |x_0 x_1 \ldots \bar{x}_{i-1} \ldots x_{n-1}\rangle := |\underline{x}^i\rangle$

$$\begin{aligned}
[\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{NAE}}, \hat{\mathcal{H}}_{\mathcal{B}}] |\underline{x}\rangle &= \hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{NAE}} \hat{\mathcal{H}}_{\mathcal{B}} |\underline{x}\rangle - \hat{\mathcal{H}}_{\mathcal{B}} \hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{NAE}} |\underline{x}\rangle \\
&= \sum_{i=0}^{n-1} \hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{NAE}} |\underline{x}^i\rangle - h^{NAE}(\underline{x}) \hat{\mathcal{H}}_{\mathcal{B}} |\underline{x}\rangle \\
&= \sum_{i=0}^{n-1} h^{NAE}(\underline{x}^i) |\underline{x}^i\rangle - h^{NAE}(\underline{x}) \sum_{i=0}^{n-1} |\underline{x}^i\rangle \\
&= \sum_{i=0}^{n-1} \left[h^{NAE}(\underline{x}^i) - h^{NAE}(\underline{x})\right] |\underline{x}^i\rangle
\end{aligned} \tag{5.47}$$

Since $|\underline{x}\rangle$ arbitrary,

$$[\hat{\mathcal{H}}_{\mathcal{C}_{\underline{\sigma}}^{NAE}}, \hat{\mathcal{H}}_{\mathcal{B}}] |\underline{x}\rangle = 0 \iff \forall \underline{x} \in \{0,1\}^n, h^{NAE}(\underline{x}^i) = h^{NAE}(\underline{x}) \tag{5.48}$$

This is clearly not true (as can be shown with a simple counterexample).

5.1.4 Classical Benchmarking

To account for the additional symmetry in k-NAE-SAT, we adapt the `WalkSATlm` algorithm (3) and introduce `WalkSATm2b2`. In particular, we did not find any dedicated k-NAE-SAT solvers in the literature. As such, this is an entirely novel contribution of this thesis.

First, we recall (2.2.2) the scoring functions for a variable x in a Boolean formula ϕ:

- $make(x)$, the number of clauses that become **satisfied** by flipping x.
- $break(x)$, the number of clauses that become **unsatisfied** by flipping x.
- $make_\tau(x)$, the number of $(\tau - 1)$-satisfied clauses that become τ-satisfied by flipping x.
- $lmake(x) = \omega_1 \cdot make_1(x) + \omega_2 \cdot make_2(x)$

where a clause c in ϕ is τ-satisfied if and only if it contains exactly τ satisfied literals. It is clear that $make_1(x) = make(x)$.

`WalkSATlm` uses the *lmake* score during tiebreaks, as such, it only considers the impact flipping a variable has on **increasing** the number of satisfied literals. This approach is relevant for k-SAT, where formulas are more likely to be satisfied if more literals are set true. On the other hand, k-NAE-SAT requires that at least one literal be unsatisfied in each clause. To take this into account, we introduce the following extensions.

Definition 5.1.2. *(NAE Make score) For a variable x in a Boolean formula ϕ, $make^{NAE}(x)$ is the number of clauses that become **satisfied** by flipping x, in the NAE formulation.*

Definition 5.1.3. *(NAE Break score) For a variable x in a Boolean formula ϕ, $break^{NAE}(x)$ is the number of clauses that become **unsatisfied** by flipping x, in the NAE formulation.*

Definition 5.1.4. *(τ-level break) For a variable x in a Boolean formula ϕ, $break_{\tau}(x)$ is the number of (τ)-satisfied clauses that become $(\tau - 1)$-satisfied by flipping x.*

Consequently, we deduce the following relationships

$$\begin{aligned} make^{NAE}(x) &= make_1(x) + break_k(x) \\ break^{NAE}(x) &= break_1(x) + make_k(x) \end{aligned} \tag{5.49}$$

These follow from the fact that in k-NAE-SAT, satisfying all the literals now means the clause is no longer satisfied. It is clear that unlike k-SAT, where $make_1(x) = make(x)$, $make_1(x) \leq make^{NAE}(x)$.

Using this, we introduce the following scoring function.

Definition 5.1.5. *(m2b2) For a variable x in a Boolean formula ϕ,*

$$m2b2(x) = \omega_1 \cdot make^{NAE}(x) + \omega_2 \cdot (make_2(x) + break_{k-1}(x)) \tag{5.50}$$

Expanding the definition using the identities in 5.49, we find

$$m2b2(x) = \omega_1 \cdot (make_1(x) + break_k(x)) + \omega_2 \cdot (make_2(x) + break_{k-1}(x)) \tag{5.51}$$

This form allows for an immediately efficient implementation of the scoring function where the algorithm keeps track of the number of true literals in each clause and considers changes caused by a variable being flipped. The `WalkSATm2b2` algorithm (4) uses $m2b2$ to resolve tiebreaks, regaining symmetry in its selection procedure.

Algorithm 4: WalkSATm2b2 Algorithm

```
1 Function WalkSATm2b2($\phi$, $p$, $max\_flips$):
2     Randomly assign truth values to all variables in $\phi$;
3     for $i = 1$ to $max\_flips$ do
          // As in WalkSATlm (3)
4         Sample $X \sim Bernoulli(p)$;
5         if $X$ then
6             Choose $x \in c$ randomly and flip its value;
7         else
8             Choose $x \in \underset{v \in B}{\operatorname{argmax}}\, m2b2(v)$, $B = \underset{y \in c}{\operatorname{argmin}}\, break(y)$, randomly and flip its value;
9     return No satisfying assignment found;
```

Intuitively, $m2b2$ rewards variables for moving clauses away from standard instability (no literals satisfied) *and* for moving away from NAE instability (all literals satisfied). A similar idea, *subscores*, is considered in the work of Cai & Su [40] where they make use of multi-level scores for k-SAT solvers. In this, however, $subbreak(x)$ - which considers clauses that move from being $(k-1)$-satisfied to $(k-2)$-satisfied - is *minimised* since this makes k-SAT clauses more unstable.

In what follows, we benchmark QAOA against both `WalkSATlm` and `WalkSATm2b2`. For both classical solvers, we carry out a grid search across

- noise $p \in \{\frac{i}{20} : 0 \leq i \leq 20\}$
- weights $\omega_1 \in \{\frac{i}{10} : 0 \leq i \leq 10\}$, $\omega_2 = 1 - \omega_1$

to identify the optimal set of hyperparameters for each k. In particular, in keeping with the $\underline{\beta}, \underline{\gamma}$ angle selection procedure for QAOA, this is done for only $t = 100$ **satisfiable** random k-NAE-SAT instances, $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(\mathbf{n = 12}, k, \hat{r}_k^{\text{NAE}})\}_{i=0}^{t-1}$.

5.2 Evaluation

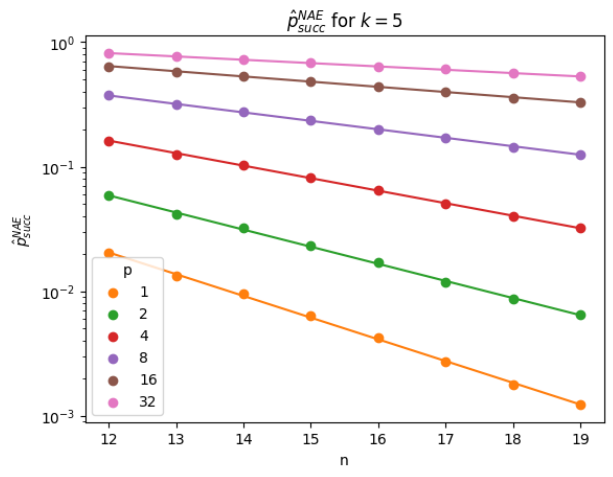
The satisfiability threshold for k-NAE-SAT is approximately half that of k-SAT (2.5). Since the number of clauses m is sampled as $m \sim Poisson(rn)$, we will analyse $k \in \{5, 9\}$ instances to parallel our discussion of $k \in \{4, 8\}$ results in the previous section. This allows us to consider instances of similar sizes. The full set of results for other k and p are listed in Appendix B.

5.2.1 Success Probabilities

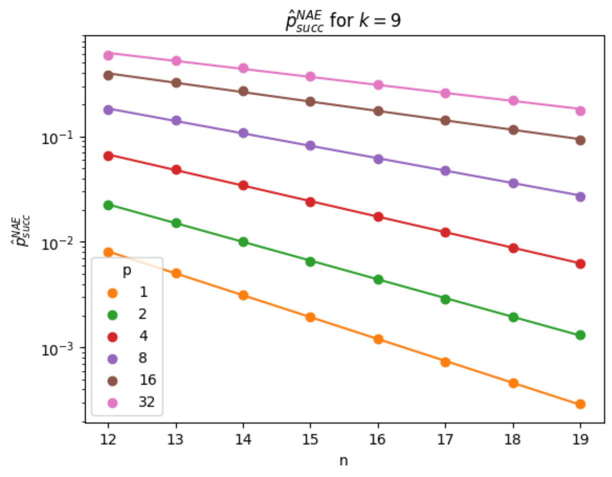
Given parameters $\underline{\beta}^*$ and $\underline{\gamma}^*$, we evaluate the success probability of QAOA over $v = 2500$ **satisfiable** random k-NAE-SAT instances, $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}_k^{\text{NAE}})\}_{i=0}^{v-1}$ and calculate

$$\hat{p}_{succ}^{\text{NAE}} = \frac{1}{v} \sum_{i=0}^{v-1} p_{succ}^{\text{NAE}}(\sigma_i) \tag{5.52}$$

This is done for $p = \{1, 2, 4, 8, 16, 32\}$ layers and $12 \leq n \leq 19$. The results (Figure 5.1) show, as expected, an exponential decay in success probability with instance size. Importantly, the rate of decay decreases as the number of layers p increases.



(a) QAOA average success probabilities across 2500 satisfiable 5-NAE-SAT $CNF(n, 5, \hat{r}_5^{\text{NAE}})$ instances (error bars too small to be seen).



(b) QAOA average success probabilities across 2500 satisfiable 9-NAE-SAT $CNF(n, 9, \hat{r}_9^{\text{NAE}})$ instances (error bars too small to be seen).

Figure 5.1: QAOA k-NAE-SAT success probabilities.

5.2.2 Running Times

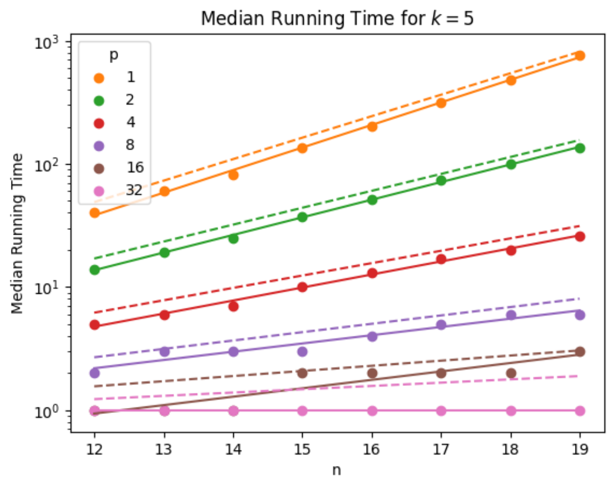
We extend the notion of (median) running times to k-NAE-SAT as follows.

Definition 5.2.1. *(Instance running time) The running time of a random k-NAE-SAT QAOA instance, $r^{NAE}_{\underline{\sigma}}$, is defined as the number of bitstrings that have to be sampled from the final quantum state $\left|\Psi^{NAE}(\underline{\beta}, \underline{\gamma}, \underline{\sigma})\right\rangle$ before one is a satisfying assignment of $\underline{\sigma}$.*

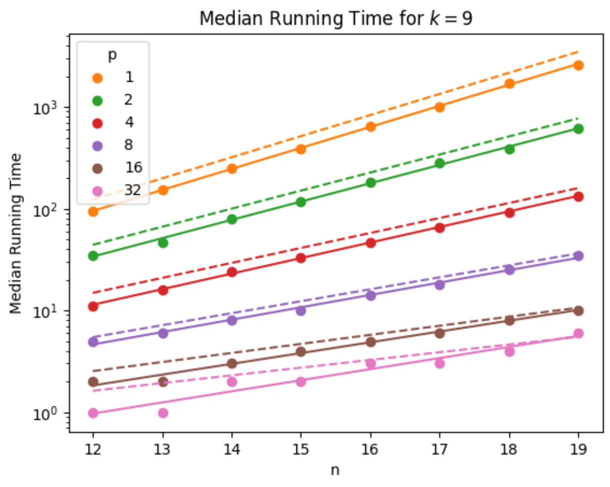
Definition 5.2.2. *(Empirical median running time) Given the ensemble $R = \{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,r)\}_{i=0}^{v-1}$, the QAOA empirical median running time $Med_{\underline{\sigma}\in R}[r^{NAE}_{\underline{\sigma}}]$ is defined as the median of the corresponding running times $\{r^{NAE}_{\underline{\sigma}_i} : \underline{\sigma}_i \in R\}$.*

Given parameters $\underline{\beta}^*$ and $\underline{\gamma}^*$, we evaluate the running time of QAOA over $v = 2500$ **satisfiable** random k-NAE-SAT instances, $R = \{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}^{\text{NAE}})\}_{i=0}^{v-1}$, and calculate $Med_{\underline{\sigma}\in R}[r^{\text{NAE}}_{\underline{\sigma}}]$.

We consider circuits with $p \in \{1, 2, 4, 8, 16, 32\}$ layers and instances where $12 \leq n \leq 19$, $k \in \{5, 9\}$. Our results (Figure 5.2) show that the median running time scales exponentially with instance size and confirm its alignment with the reciprocal of the success probability. Importantly, the slope of scaling decreases as the number of layers p increases.



(a) QAOA median running times across 2500 satisfiable 5-NAE-SAT $CNF(n, 5, \hat{r}_5^{\text{NAE}})$ instances. Dashed lines are the reciprocal of corresponding success probabilities.



(b) QAOA median running times across 2500 satisfiable 9-NAE-SAT $CNF(n, 9, \hat{r}_9^{\text{NAE}})$ instances. Dashed lines are the reciprocal of corresponding success probabilities.

Figure 5.2: QAOA k-NAE-SAT median running times.

Observing that the two metrics agree more strongly for $k = 9$ than $k = 5$, we hypothesise that this is related to the discrete nature of the running time in contrast to the continuity of the success probability. In particular, since a smaller k corresponds to generally less challenging instances, the QAOA running times are similar across instance sizes and produce flat scalings. A similar effect occurs for larger values of p. In this case, we expect the QAOA circuits with more layers to be more powerful, requiring fewer samples to extract a satisfying assignment.

We investigate our hypothesis further by assessing the exponents induced by the two metrics. In particular, recalling the previously introduced success probability and median running time scaling exponents (3.1.3.3), we extend them to k-NAE-SAT as follows.

Definition 5.2.3. *(NAE Empirical success probability scaling exponent) Given the ensemble $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,r)\}_{i=0}^{v-1}$, $\hat{C}^{NAE}_{p,k}(\underline{\beta}, \underline{\gamma})$ is such that*

$$\hat{p}^{NAE}_{succ} = \frac{1}{v}\sum_{i=0}^{v-1} p^{NAE}_{succ}(\underline{\sigma}_i) = 2^{-n\hat{C}^{NAE}_{p,k}(\underline{\beta},\underline{\gamma})} \tag{5.53}$$

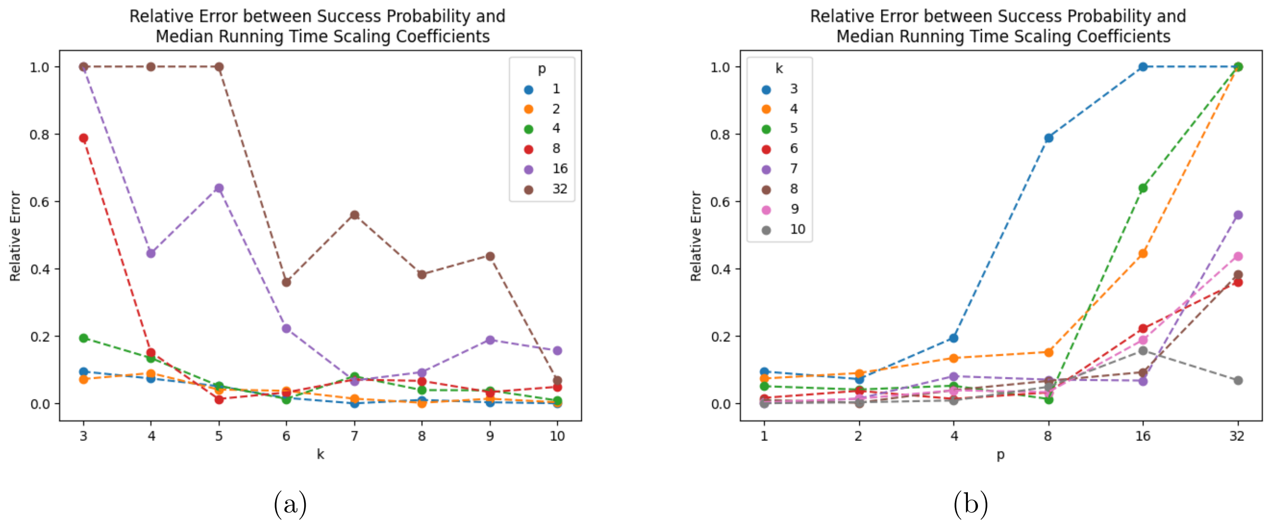
Definition 5.2.4. *(NAE Empirical median running time scaling exponent) Given the ensemble* $R = \{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,r)\}_{i=0}^{v-1}$, $\tilde{C}^{NAE}_{p,k}(\underline{\beta},\underline{\gamma})$ *is such that*

$$Med_{\underline{\sigma}\in R}[r^{NAE}_{\underline{\sigma}}] = 2^{n\tilde{C}^{NAE}_{p,k}(\underline{\beta},\underline{\gamma})} \tag{5.54}$$

We consider the relative error of these exponents across $3 \le k \le 10$, defined as

$$\left| \frac{\hat{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma}) - \tilde{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma})}{\hat{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma})} \right| \tag{5.55}$$

The results (Figure 5.3) support our observations: we find that the error decreases as k increases and increases as p increases. This is particularly clear in Figure 5.3c where we consider more values of p.



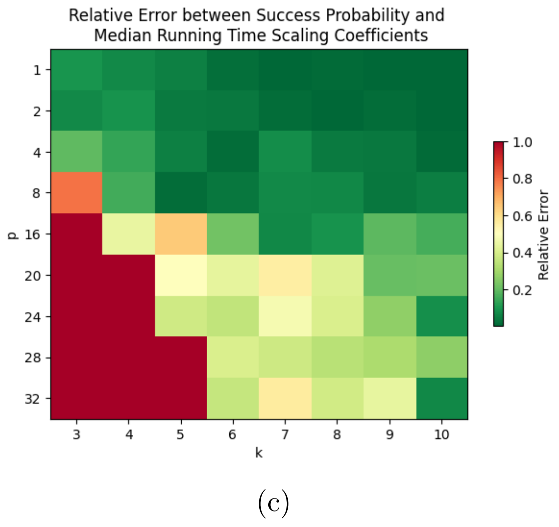


Figure 5.3: $\hat{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma})$, $\tilde{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma})$ relative error across 2500 satisfiable k-NAE-SAT $CNF(n,k,\hat{r}^{\text{NAE}}_k)$ instances.

We note that these effects are also in part due to the relatively small values of n which we are considering. We do not expect that the median running times will continue to scale as flatly in the large n limit, even for the large p that we have considered. As such, it is important to consider the scaling of our exponents as a function of p to take these effects into account. We consider this in what follows.

5.2.3 Benchmarking

We compare the median running time of QAOA to that of `WalkSATlm` and `WalkSATm2b2` across 2500 **satisfiable** random k-NAE-SAT instances, $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n,k,\hat{r}^{\text{NAE}}_k\}_{i=0}^{v-1}$, for $k \in \{5,9\}, 12 \le n \le$

19. As for k-SAT, we define and justify (4.2.3) the runtime of both classical solvers as the number of loop iterations made.

We observe (Figures 5.4a, 5.4b) that the median running time of QAOA outperforms `WalkSATm2b2` which in turn outperforms `WalkSATlm` for both $k = 5$ and $k = 9$. Further, we assess the scaling of these runtimes rather than their absolute values (Figure 5.4c) and find that for both $k = 5$ and $k = 9$, there exists a threshold p above which QAOA's scaling improves on `WalkSATm2b2`. We fit the coefficients to a power law $\sim ap^b$ and calculate that the coefficients scale as

$$\tilde{C}^{\text{NAE}}_{p,5}(\underline{\beta}, \underline{\gamma}) \sim 0.64p^{-0.52} \quad (5.56) \qquad \tilde{C}^{\text{NAE}}_{p,9}(\underline{\beta}, \underline{\gamma}) \sim 0.69p^{-0.22} \quad (5.57)$$

The power law fit for $k = 5$ begins to disagree with the coefficients in the large p limit. As in the previous section, this is likely due to the fact that the instances are easy for QAOA to solve, regardless of the problem size n (for the range that we considered here). As such, effectively no scaling is induced since the running times are generally ≈ 1. It is clear that this is no longer the case for $k = 9$ where we find improved agreement due to harder problem instances.

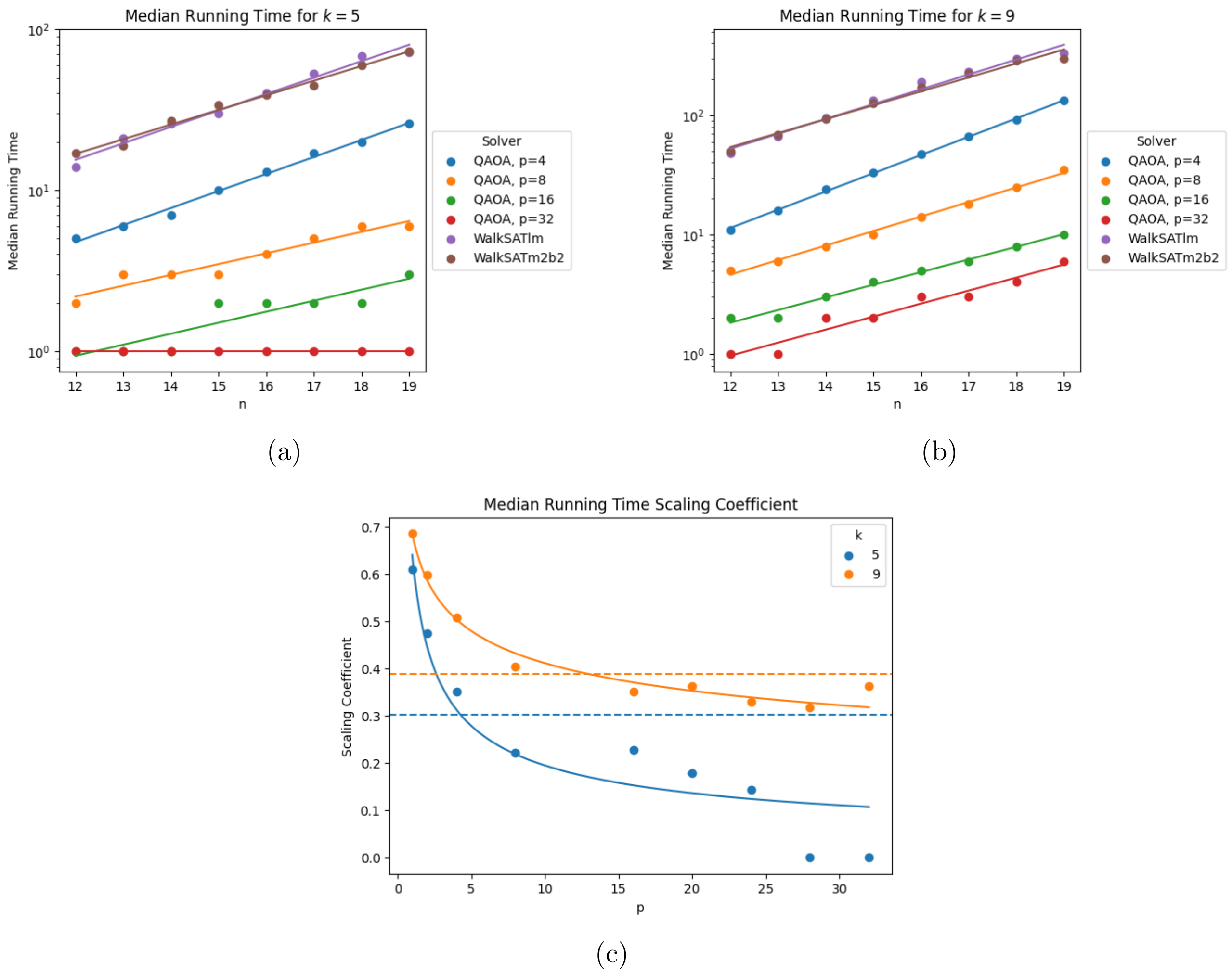


Figure 5.4: (5.4a, 5.4b) Median running times across 2500 satisfiable k-NAE-SAT $CNF(n, k, \hat{r}^{\text{NAE}}_k)$ instances of QAOA, `WalkSATlm` and `WalkSATm2b2`. (5.4c) Induced scaling of QAOA k-NAE-SAT median running times $\tilde{C}^{\text{NAE}}_{p,k}(\underline{\beta}, \underline{\gamma})$. Dashed lines are observed `WalkSATm2b2` scalings.

5.2.4 Excessive Scaling

As for k-SAT, we examine the notion of *excessive scaling* (4.2.4) by considering the scaling induced by QAOA in cases where the initial state $|+\rangle^{\otimes n}$ is mostly unchanged by the circuit. First, we extend the definition of the predicted scaling exponent (3.17).

Definition 5.2.5. *(NAE Predicted scaling exponent)*

$$C^{NAE}_{p,k}(\underline{\beta},\underline{\gamma}) = -\lim_{n\to\infty}\frac{1}{n}\log \mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p^{NAE}_{succ}(\underline{\sigma})] \tag{5.58}$$

Theorem 5.2.1. *(k-NAE-SAT QAOA random assignment scaling exponent)*

$$C^{NAE}_{p,k}(\underline{\beta},\underline{0}) = C^{NAE}_{p,k}(\underline{0},\underline{\gamma}) = 2^{1-k}r \tag{5.59}$$

Proof. First, we realise $\forall p$

$$\begin{cases} \left|\Psi^{\text{NAE}}(\underline{\beta},\underline{0},\underline{\sigma})\right\rangle_p = \prod_{i=0}^{p-1}\hat{\mathcal{U}}_{\mathcal{B}}(\beta_i)\left|+\right\rangle^{\otimes n} & \propto \left|+\right\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}}\sum_{\underline{x}\in\{0,1\}^n}\left|\underline{x}\right\rangle \\ \left|\Psi^{\text{NAE}}(\underline{0},\underline{\gamma},\underline{\sigma})\right\rangle_p = \prod_{i=0}^{p-1}\hat{\mathcal{U}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}}(\gamma_i)\left|+\right\rangle^{\otimes n} & \propto \left|+\right\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}}\sum_{\underline{x}\in\{0,1\}^n}\left|\underline{x}\right\rangle \end{cases} \tag{5.60}$$

since in both cases, the resulting operators all commute and the state is simply effected with a global phase (2.49). As such,

$$\begin{aligned} p^{\text{NAE}}_{succ}(\underline{\sigma}) &= \frac{1}{2^n}\sum_{\underline{x},\underline{x}'\in\{0,1\}^n}\left\langle\underline{x}'\right|\{\hat{\mathcal{H}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}}=0\}\left|\underline{x}\right\rangle \\ &= \frac{1}{2^n}\sum_{\underline{x},\underline{x}'\in\{0,1\}^n}\Pr(\underline{x}\vdash_{\text{NAE}}\underline{\sigma})\delta_{\underline{x},\underline{x}'} \\ &= \frac{1}{2^n}\sum_{\underline{x}\in\{0,1\}^n}\Pr(\underline{x}\vdash_{\text{NAE}}\underline{\sigma}) \end{aligned} \tag{5.61}$$

Since $\{\hat{\mathcal{H}}_{\mathcal{C}^{\text{NAE}}_{\underline{\sigma}}}=0\}$ denotes the orthogonal projector onto the space of satisfying assignments.

As $\underline{\sigma}$ is in CNF

$$\Pr(\underline{x}\vdash_{\text{NAE}}\underline{\sigma}) = \prod_{i=0}^{m-1}\Pr(\underline{x}\vdash_{\text{NAE}}\sigma_i) = \prod_{i=0}^{m-1}\left[1-\Pr(\underline{x}\nvdash_{\text{NAE}}\sigma_i)\right] \tag{5.62}$$

Recalling 5.6:

$$\Pr(\underline{x}\nvdash_{\text{NAE}}\sigma_i) = \Pr\left(\underline{x}\nvdash\sigma_i \vee \underline{x}\vdash\bigwedge_{j=0}^{k-1}l_{\sigma_{ij}}\right) = \left(\frac{1}{2}\right)^k + \left(\frac{1}{2}\right)^k = 2^{1-k} \tag{5.63}$$

since all literals are chosen independently and satisfying a literal occurs with probability 1/2 as it is negated with equal probability.

As such,

$$p^{\text{NAE}}_{succ}(\underline{\sigma}) = \frac{1}{2^n}\sum_{\underline{x}\in\{0,1\}^n}\prod_{i=0}^{m-1}\left[1-\Pr(\underline{x}\nvdash_{\text{NAE}}\sigma_i)\right] = \frac{1}{2^n}\sum_{\underline{x}\in\{0,1\}^n}\left[1-2^{1-k}\right]^{m(\underline{\sigma})} \tag{5.64}$$

since all the clauses are chosen independently. We write $m(\underline{\sigma})$ to emphasise m is a property of the random instance $\underline{\sigma}$.

Finally, as $m \sim Poisson(rn)$

$$\begin{aligned}
\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}\left[\left(1-2^{1-k}\right)^{m(\underline{\sigma})}\right] &= \mathbb{E}_{m\sim Poisson(nr)}\left[\left(1-2^{1-k}\right)^{m}\right] \\
&= \sum_{m\geq 0}\frac{e^{-rn}(rn)^m}{m!}(1-2^{1-k})^m \\
&= e^{-rn}\sum_{m\geq 0}\frac{\left[rn(1-2^{1-k})\right]^m}{m!} \\
&= e^{-rn}e^{rn(1-2^{1-k})} \\
&= e^{-rn2^{1-k}}
\end{aligned} \tag{5.65}$$

This means that

$$\lim_{n\to\infty}\frac{1}{n}\log\mathbb{E}_{\underline{\sigma}\sim CNF(n,k,r)}[p^{\text{NAE}}_{succ}(\underline{\sigma})] = \lim_{n\to\infty}\frac{1}{n}\log e^{-rn2^{1-k}} = -2^{1-k}r \tag{5.66}$$

and so

$$C^{\text{NAE}}_{p,k}(\underline{\beta},\underline{0}) = C^{\text{NAE}}_{p,k}(\underline{0},\underline{\gamma}) = 2^{1-k}r \tag{5.67}$$

□

In the limit $v\to\infty$ (5.54), we expect $\hat{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma}) = C^{\text{NAE}}_{p,k}(\underline{\beta},\underline{0})$. Further, we have studied the relationship between $\hat{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma})$ and $\tilde{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma})$ and found that they agree well. As such, we compare $\tilde{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma}) = 2^{1-k}r$ to the empirical observations in our experiments. As anticipated, QAOA outperforms the random scaling (Figure 5.5). Interestingly, as in k-SAT, in the small p regime, the scaling is effectively that of random assignment. This is due to the fact that few-layer circuits are inexpressive leading to outputs that are effectively unchanged from the initial state.

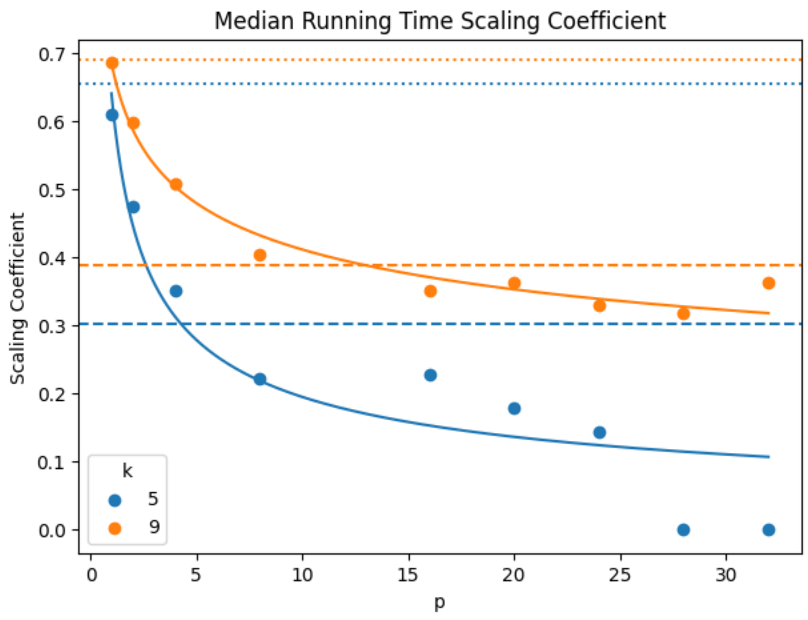


Figure 5.5: Induced scaling of QAOA k-NAE-SAT median running times $\tilde{C}^{\text{NAE}}_{p,k}(\underline{\beta},\underline{\gamma})$. Dashed line is observed `WalkSATm2b2` scaling. Dotted line is random assignment scaling.

5.2.5 Discussion

The results above provide three main contributions, in addition to the QAOA encodings derived for k-NAE-SAT.

First, we have shown that the success probability agrees strongly with the median running time. This suggests that, as for k-SAT, we can make use of an analytic heuristic to predict the empirical performance of QAOA, over an ensemble of random instances. This is particularly useful in the current NISQ-era, where we only have access to quantum processors with low gate fidelities and high qubit decoherence rates [41]. Meanwhile, classical simulations are limited to less than 50 qubits [42] so don't provide a practical means of predicting the large n QAOA performance. We explored the strength of agreement between the success probability and median running time and identified regimes where they diverge. However, these were primarily related to problem instances being too easy or the QAOA circuits too powerful - both situations of lesser interest for the purposes of solving difficult problems.

We have introduced a novel stochastic local search (2.2) algorithm for k-NAE-SAT, `WalkSATm2b2`, and shown that it outperforms its predecessor `WalkSATlm`. This identifies that the symmetries of the problem can be exploited to improve running time. Benchmarking QAOA against both solvers, we find a quantum advantage across all k-NAE-SAT instances of $3 \leq k \leq 9$. Importantly, all solver hyperparameters were fine-tuned over the same set of problem instances, suggesting that QAOA is more capable of generalising. The advantage occurs at a threshold number of layers $p \approx 5$ for $k = 5$ and $p \approx 14$ for $k = 9$. These are similar thresholds to those found for 4-SAT and 8-SAT, and suggest, as anticipated, that the performance of QAOA depends on the size of the problem instance being considered. Namely, we considered instances with clause densities near the satisfiability threshold, which scales as 2^k for k-SAT and 2^{k-1} for k-NAE-SAT. As such, we predicted, and found, that the results for k-SAT and $(k+1)$-NAE-SAT were similar.

Finally, we considered the excessive scaling of QAOA for k-NAE-SAT and showed that for small p, QAOA circuits act closely to a random assignment algorithm. This confirms the inexpressivity of shallow depth circuits. Nonetheless, the scaling rapidly decreases for a small increase in the number of layers, highlighting the effectiveness of the QAOA ansatz and suggesting that large circuits are not necessary for advantages over classical algorithms.

Chapter 6

Conclusions and Future Work

In this work, we successfully:

- Derived novel encodings of k-SAT and k-NAE-SAT for QAOA that could be run on any gate-based quantum computer and implemented them in `Qiskit` [13]. This was done with an object oriented approach, allowing general combinatorial optimisation problems to be considered.
- Studied the cost of classically simulating our encodings and introduced *diagonalisations* of our Hamiltonians to reduce the effective time complexity from $\mathcal{O}\big(2^k k^3 N^2 \log N\big)$ to $\mathcal{O}(N)$, for a system size of $N = 2^n$. This was experimentally improved with preprocessing and parallelisation.
- Reproduced the results of Boulebnane & Montanaro [10], implementing an efficient simulator in `PyTorch` [14], and confirmed the quantum advantage of QAOA for 8-SAT over the best performing classical solver `WalkSATlm`.
- Produced novel *success probability* and *median running time* results of QAOA for 4-SAT and a large range of k-NAE-SAT problems. We highlighted regimes where the scalings disagreed, yet showed that these were largely in situations where problem instances were easy and so less important. We also considered the scaling of these heuristics in the large p limit and interpreted their *excessive scaling* performance.
- Introduced `WalkSATm2b2`, an extension of `WalkSATlm` that accounted for the symmetry of k-NAE-SAT instances and showed that it improved performance. Yet, we showed that QAOA outperformed both solvers on a range of k-NAE-SAT instances.

A number of extensions naturally arise from the work achieved in this project. We consider them as follows.

Extensions of empirical findings. While we considered a range of problem instances $CNF(n, k, r)$ and QAOA circuits, further assessment should be done in other regimes. In particular:

- Problems with other clause densities r, including the unique solution threshold and the clustering threshold (solutions can be organised into clusters where they only differ by one variable assignment) [12].
- Problems with other variable numbers n and k. We note that these will require large efforts in simulation efficiency to allow for realistic run times - both for the classical and quantum solvers.
- Evaluation over more problem instances. We only considered 2500 instances to evaluate the solvers. Nonetheless, errors were found to be insignificant and unnoticeable.
- QAOA circuits with more fine-grained and larger layer counts p. Again, this will require engineering efforts.

Theoretical analysis. While our empirical findings provide a solid indication of QAOAs performance for k-NAE-SAT, this can be improved through a theoretical understanding. This is particularly relevant to the study of large instance sizes $n \to \infty$, where it is impractical to simulate circuits with such large widths. This can involve the techniques of *multinomial sums*, introduced by Boulebnane & Montanaro [10].

NAE-SAT classical solvers. We introduced `WalkSATm2b2` and showed it outperformed `WalkSATlm` on k-NAE-SAT instances. Further extensions should be considered as well as entirely new classical solvers. It is important to identify regimes where there is *true* quantum advantage. As such, it is equally important to identify possible improvements to current classical approaches.

Variants of SAT and NAE-SAT QAOA. In this work, we focused on *fixed angle* QAOA, selecting hyperparameters with one procedure. While we demonstrated that this was sufficient to identify a quantum advantage, other procedures may improve QAOA's ability to generalise and reduce the number of layers p required. Possible extensions can be categorised as:

1. Non-fixed angle QAOA, where parameters are trained on the instance itself. This is computationally expensive and so the incurred costs should be studied and justified over a generalised procedure.

2. Other procedures for *directly* generating hyperparameters. Again, this should take into consideration the cost of the procedure and not just improvements to the success probability of QAOA.

3. Other procedures of *indirectly* generating hyperparameters. This can include warm starts, where the parameters for other k and p are used as the initial values in training. Qualitatively, we found that there was often agreement of $\underline{\beta}^*$ and $\underline{\gamma}^*$ in regimes of similar problems and layer counts.

Variants of SAT. This work focused on k-SAT and k-NAE-SAT. The potential of QAOA can be studied for other problems as both an exact (1-in-k-SAT, Graph-k-colouring, etc) and approximate (MAX-k-SAT, MAX-k-CUT, etc) solver. Our object oriented approach allows for a straightforward extension to such problems.

Appendix A

General Representations of Boolean and Real Functions

A.1 Boolean Functions

Theorem A.1.1. *Let $f \in \mathcal{B}_n$. The unique Hamiltonian $\hat{\mathcal{H}}_f$ representing f is [28]:*

$$\hat{\mathcal{H}}_f = \sum_{S \subset [n]} \hat{f}(S) \prod_{j \in S} Z_j = \hat{f}(\emptyset) I + \sum_{j=1}^{n} \hat{f}(\{j\}) Z_j + \sum_{j<k} \hat{f}(\{j,k\}) Z_j Z_k \dots \tag{A.1}$$

with Fourier coefficients:

$$\hat{f}(S) = \frac{1}{2^n} \sum_{\underline{x} \in \{0,1\}^n} f(\underline{x})(-1)^{S \cdot \underline{x}} = \frac{1}{2^n} tr(\hat{\mathcal{H}}_f \prod_{j \in S} Z_j) \tag{A.2}$$

where the following notation has been used:

- $S \cdot \underline{x} := \sum_{j \in S} x_j$
- $[n] := \{1, 2, \dots, n\}$
- $Z_j := I^{\otimes(j-1)} \otimes Z \otimes I^{\otimes(n-j)}$ *i.e. the Z-Pauli operator applied to the j^{th} qubit.*

Example A.1.1. *Let $f : \{0,1\}^3 \to \{0,1\}, f(\underline{x}) = x_1 \oplus x_2 \oplus x_3$*

$$\begin{aligned}
\hat{f}(\emptyset) &= \frac{1}{8} \sum_{\underline{x} \in \{0,1\}^3} f(\underline{x})(-1)^{\emptyset \cdot \underline{x}} = \frac{1}{8} \sum_{\underline{x} \in \{0,1\}^3} f(\underline{x}) = \frac{1}{2} \\
\hat{f}(\{j\}) &= \frac{1}{8} \sum_{\underline{x} \in \{0,1\}^3} f(\underline{x})(-1)^{\{j\} \cdot \underline{x}} = \frac{1}{8} \sum_{\underline{x} \in \{0,1\}^3} f(\underline{x})(-1)^{x_j} = 0 \\
\hat{f}(\{j,k\}) &= \frac{1}{8} \sum_{\underline{x} \in \{0,1\}^3} f(\underline{x})(-1)^{\{j,k\} \cdot \underline{x}} = \frac{1}{8} \sum_{\underline{x} \in \{0,1\}^3} f(\underline{x})(-1)^{x_j + x_k} = 0 \\
\hat{f}(\{j,k,l\}) &= \frac{1}{8} \sum_{\underline{x} \in \{0,1\}^3} f(\underline{x})(-1)^{\{j,k,l\} \cdot \underline{x}} = \frac{1}{8} \sum_{\underline{x} \in \{0,1\}^3} f(\underline{x})(-1)^{x_j + x_k + x_l} = -\frac{1}{2}
\end{aligned} \tag{A.3}$$

The representing Hamiltonian is thus

$$\hat{\mathcal{H}}_f = \sum_{S \subset [3]} \hat{f}(S) \prod_{j \in S} Z_j = \frac{1}{2}(I - Z_1 Z_2 Z_3) \tag{A.4}$$

We verify its application

$$\hat{\mathcal{H}}_f |111\rangle = \frac{1}{2}(I - Z_1 Z_2 Z_3) = \frac{1}{2}(1 - (-1)) |111\rangle = 1 \times |111\rangle \tag{A.5}$$

as expected.

A.2 Real/Pseudo-Boolean Functions

Theorem A.2.1. *For $g \in \mathcal{R}_n$, the unique Hamiltonian representing g is $\hat{\mathcal{H}}_g$ s.t* [28]*:*

$$\hat{H}_g = \sum_{S \subset [n]} \hat{g}(S) \prod_{j \in S} Z_j \tag{A.6}$$

with Fourier coefficients:

$$\hat{g}(S) = \frac{1}{2^n} \sum_{\underline{x} \in \{0,1\}^n} g(\underline{x})(-1)^{S \cdot \underline{x}} = \frac{1}{2^n} tr(\hat{H}_g \prod_{j \in S} Z_j) \tag{A.7}$$

where the following notation has been used:

- $S \cdot \underline{x} := \sum_{j \in S} x_j$
- $[n] := \{1, 2, \ldots, n\}$
- $Z_j := I^{\otimes(j-1)} \otimes Z \otimes I^{\otimes(n-j)}$ *i.e. the Z-Pauli operator applied to the j^{th} qubit.*

Appendix B

Complete Results for k-NAE-SAT

B.1 Success Probability

Given parameters $\underline{\beta}^*$ and $\underline{\gamma}^*$, we evaluate the success probability of QAOA over $v = 2500$ **satisfiable** random k-NAE-SAT instances, $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}_k^{\text{NAE}})\}_{i=0}^{v-1}$ and calculate

$$\hat{p}_{succ}^{\text{NAE}} = \frac{1}{v}\sum_{i=0}^{v-1} p_{succ}^{\text{NAE}}(\sigma_i) \tag{B.1}$$

The results for $3 \leq k \leq 10$ and $p \in \{1, 2, 4, 8, 16, 20, 24, 28, 32\}$ are as in Figure B.1.

B.2 Median Running Times

Given parameters $\underline{\beta}^*$ and $\underline{\gamma}^*$, we evaluate the running time of QAOA over $v = 2500$ **satisfiable** random k-NAE-SAT instances, $r = \{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}^{\text{NAE}})\}_{i=0}^{v-1}$, and calculate the median

$$Med_{\underline{\sigma} \in r}[r_{\underline{\sigma}}^{\text{NAE}}] \tag{B.2}$$

The results for $3 \leq k \leq 10$ and $p \in \{1, 2, 4, 8, 16, 20, 24, 28, 32\}$ are as in Figure B.2 where we compare the median running time to the reciprocal of the sucess probability.

B.3 Benchmarking

We compare the median running time of QAOA to that of `WalkSATlm` and `WalkSATm2b2` across 2500 **satisfiable** random k-NAE-SAT instances, $\{\underline{\sigma}_i : \underline{\sigma}_i \sim CNF(n, k, \hat{r}_k^{\text{NAE}}\}_{i=0}^{v-1}$.

The results for $3 \leq k \leq 9$ and $p \in \{1, 2, 4, 8, 16, 20, 24, 28, 32\}$ are as in Figure B.3. [1]

B.4 Excessive Scaling

We assess the scaling $\tilde{C}_{p,k}^{\text{NAE}}(\underline{\beta}, \underline{\gamma})$ of the above runtimes and fit the points to a power law $\sim ap^b$. This is compared to random assignment scaling $\tilde{C}_{p,k}^{\text{NAE}}(\underline{\beta}, \underline{\gamma}) = 2^{1-k}r$.

The results for $3 \leq k \leq 9$ and $p \in \{1, 2, 4, 8, 16, 20, 24, 28, 32\}$ are as in Figure B.4 and Table B.1.

[1] Due to time and resource constraints we did not benchmark for $k = 10$.

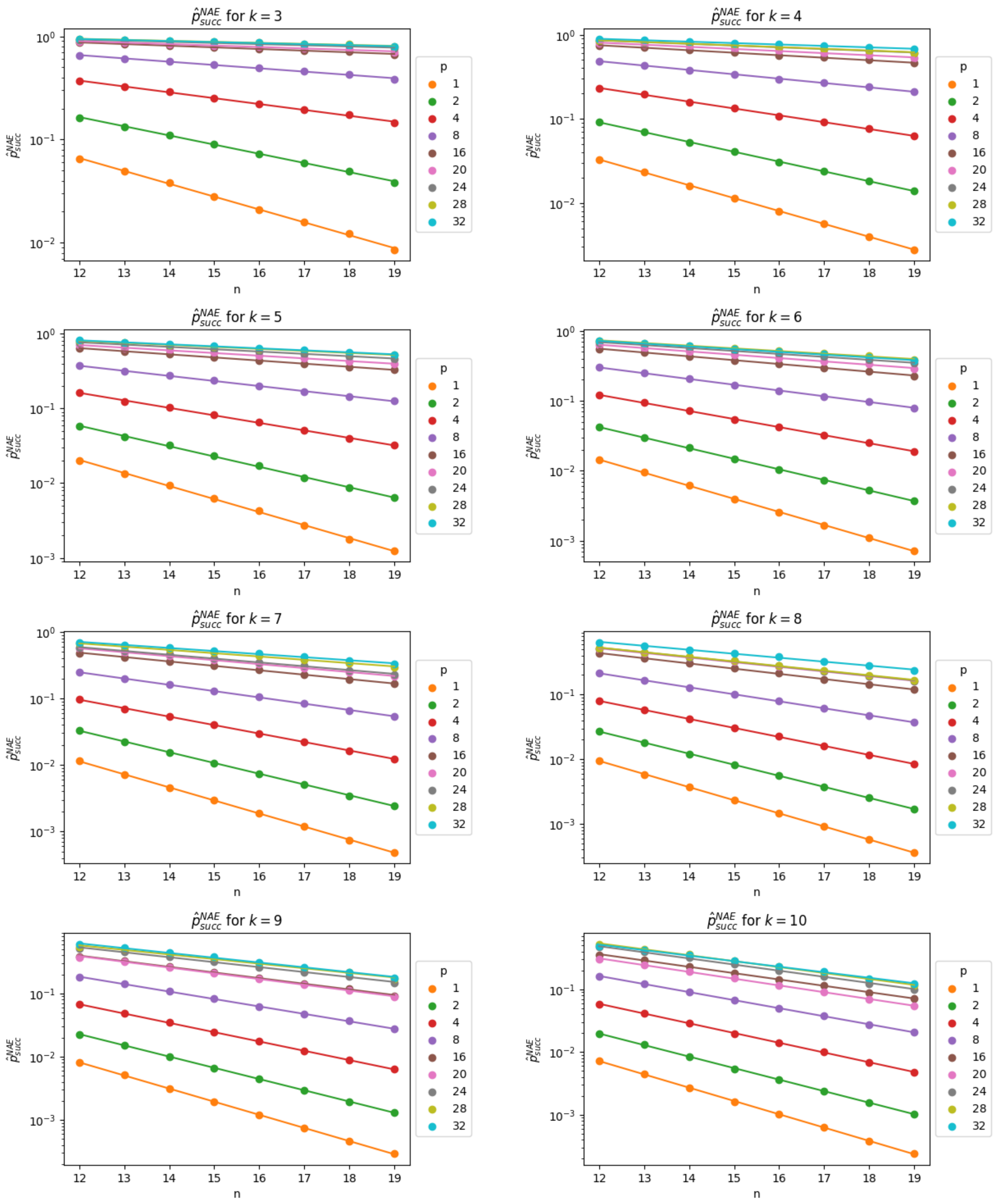


Figure B.1: QAOA average success probabilities across 2500 satisfiable k-NAE-SAT $CNF(n, k, \hat{r}_k^{\mathrm{NAE}})$ instances (error bars too small to be seen).

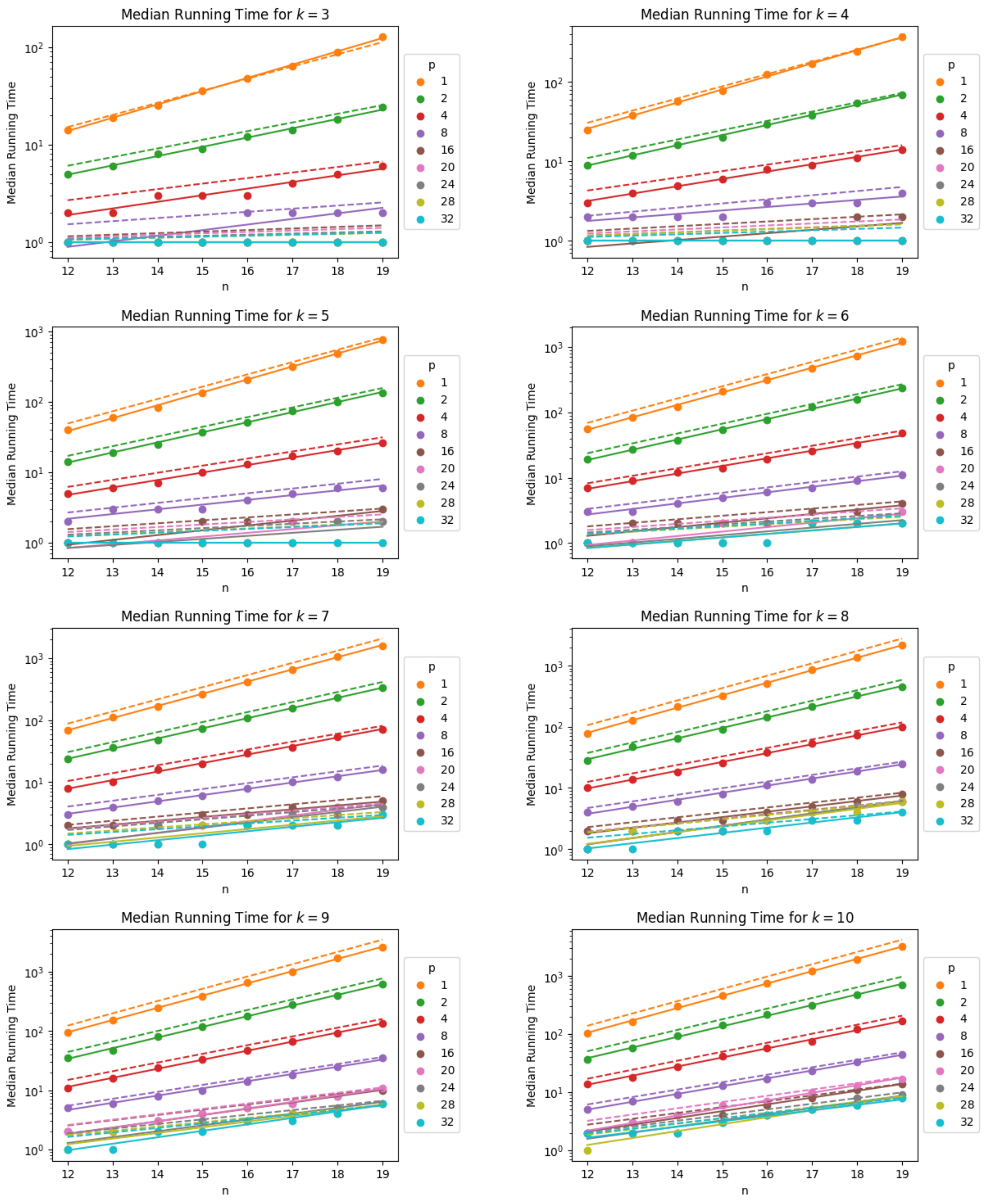


Figure B.2: QAOA median running times across 2500 satisfiable k-NAE-SAT $CNF(n, k, \hat{r}_k^{\mathrm{NAE}})$ instances. Dashed lines are the reciprocal of corresponding success probabilities.

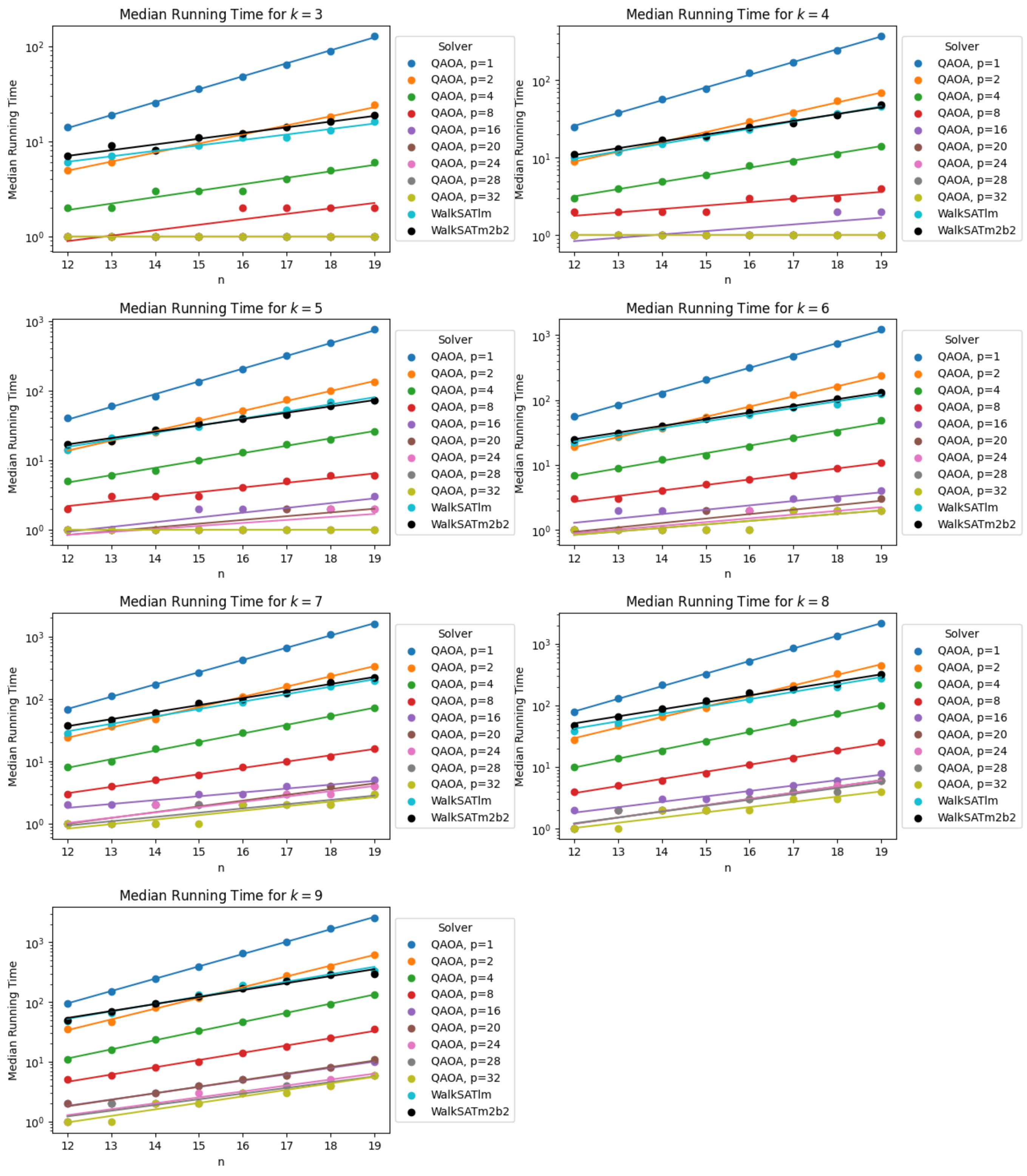


Figure B.3: Median running times across 2500 satisfiable k-NAE-SAT $CNF(n, k, \hat{r}_k^{\mathrm{NAE}})$ instances of QAOA, `WalkSATlm` and `WalkSATm2b2`.

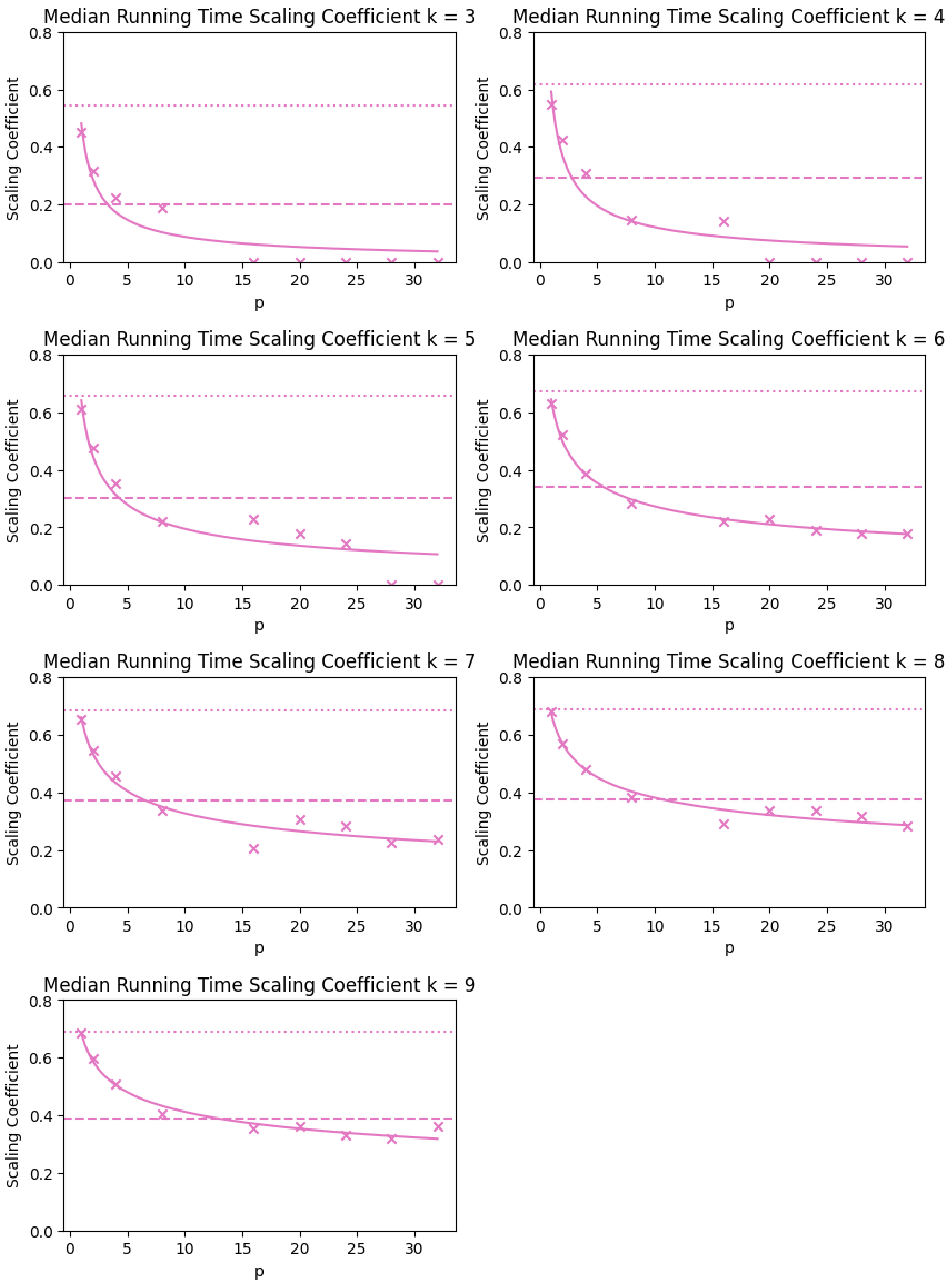


Figure B.4: Induced scaling of QAOA k-NAE-SAT median running times. Dashed line is observed `WalkSATm2b2` scaling. Dotted line is random assignment scaling.

k	a	b
3	0.48193188	-0.73653218
4	0.59220998	-0.68702534
5	0.64106651	-0.51678505
6	0.64427816	-0.37302963
7	0.66099965	-0.30455395
8	0.67472907	-0.24718223
9	0.68581328	-0.22158058
10	0.6987176	-0.19971659

Table B.1: Power law fit $\tilde{C}^{\mathrm{NAE}}_{p,k}(\underline{\beta}, \underline{\gamma}) \sim ap^b$.

Bibliography

[1] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. Available from https://doi.org/10.1145/800157.805047.

[2] Lintao Zhang and Sharad Malik. The quest for efficient boolean satisfiability solvers. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, page 17–36, Berlin, Heidelberg, 2002. Springer-Verlag. Available from https://www.princeton.edu/~chaff/publication/cade_cav_2002.pdf.

[3] Ted Hong, Yanjing Li, Sung-Boem Park, Diana Mui, David Lin, Ziyad Abdel Kaleq, Nagib Hakim, Helia Naeimi, Donald S. Gardner, and Subhasish Mitra. Qed: Quick error detection tests for effective post-silicon validation. In *2010 IEEE International Test Conference*, pages 1–10, 2010. Available from https://ieeexplore.ieee.org/document/5699215.

[4] Henry A. Kautz and Bart Selman. Planning as satisfiability. In Bernd Neumann, editor, *10th European Conference on Artificial Intelligence, ECAI 92, Vienna, Austria, August 3-7, 1992. Proceedings*, pages 359–363. John Wiley and Sons, 1992. Available from https://www.cs.cornell.edu/selman/papers/pdf/92.ecai.satplan.pdf.

[5] Daniel Jackson and Mandana Vaziri. Finding bugs with a constraint solver. In *Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '00, page 14–25, New York, NY, USA, 2000. Association for Computing Machinery. Available from https://doi.org/10.1145/347324.383378.

[6] Cristopher Moore and Stephan Mertens. *The Nature of Computation*. Oxford University Press, 2011. Available from https://www.amazon.co.uk/Nature-Computation-Cristopher-Moore/dp/0199233217.

[7] Amin Coja-Oghlan. Random constraint satisfaction problems. In S. Barry Cooper and Vincent Danos, editors, *Proceedings Fifth Workshop on Developments in Computational Models–Computational Models From Nature, DCM 2009, Rhodes, Greece, 11th July 2009*, volume 9 of *EPTCS*, pages 32–37, 2009. Available from https://doi.org/10.4204/EPTCS.9.4.

[8] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014. Available from https://arxiv.org/abs/1411.4028.

[9] Ruslan Shaydulin and Yuri Alexeev. Evaluating quantum approximate optimization algorithm: A case study. In *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*. IEEE, oct 2019. Available from https://doi.org/10.1109%2Figsc48788.2019.8957201.

[10] Sami Boulebnane and Ashley Montanaro. Solving boolean satisfiability problems with the quantum approximate optimization algorithm, 2022. Available from https://arxiv.org/abs/2208.06909.

[11] Shaowei Cai, Chuan Luo, and Kaile Su. Improving WalkSAT By Effective Tie-Breaking and Efficient Implementation. *The Computer Journal*, 58(11):2864–2875, 11 2014. Available from https://doi.org/10.1093/comjnl/bxu135.

[12] Allan Sly, Nike Sun, and Yumeng Zhang. The number of solutions for random regular nae-sat, 2016. Available from https://arxiv.org/abs/1604.08546.

[13] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023. Available from https://doi.org/10.5281/zenodo.2573505.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. Available from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[15] Michael Huth and Mark Ryan. *Logic in computer science : modelling and reasoning about systems*. Cambridge University Press, Cambridge [U.K.]; New York, 2004. Available from http://www.amazon.de/s/url=search-alias%3Daps&field-keywords=052154310X.

[16] Wikipedia. Petersen graph 3-coloring, 2006. Available from https://commons.wikimedia.org/wiki/File:Petersen_graph_3-coloring.svg. Accessed: June 1, 2023.

[17] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Journal of Symbolic Logic*, 32(1):118–118, 1967. Available from https://doi.org/10.2307/2271269.

[18] Bart Selman, Henry Kautz, and Bram Cohen. Noise strategies for improving local search. *Proceedings of the National Conference on Artificial Intelligence*, 1, 09 1999. Available from https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=EA15D3B4A4BA9334625A14FC12A28742?doi=10.1.1.319.7660&rep=rep1&type=pdf.

[19] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution, 2000. Available from https://arxiv.org/abs/quant-ph/0001106.

[20] A. Messiah. *Quantum Mechanics*. Number v. 2 in Quantum Mechanics. Elsevier Science, 1961. Available from https://books.google.co.uk/books?id=VR93vUk8d_8C.

[21] M. Born and V. Fock. Beweis des Adiabatensatzes. *Zeitschrift fur Physik*, 51(3-4):165–180, March 1928. Available from https://doi.org/10.1007/BF01343193.

[22] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Rev. Mod. Phys.*, 90:015002, Jan 2018. Available from https://link.aps.org/doi/10.1103/RevModPhys.90.015002.

[23] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Daniel Nagaj. How to make the quantum adiabatic algorithm fail, 2005. Available from https://arxiv.org/abs/quant-ph/0512159.

[24] Naomichi Hatano and Masuo Suzuki. Finding exponential product formulas of higher orders. In *Quantum Annealing and Other Optimization Methods*, pages 37–68. Springer Berlin Heidelberg, nov 2005. Available from https://doi.org/10.1007%2F11526216_2.

[25] Yin Sun, Jun-Yi Zhang, Mark S. Byrd, and Lian-Ao Wu. Adiabatic quantum simulation using trotterization, 2018. Available from https://arxiv.org/abs/1805.11568.

[26] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), jul 2014. Available from https://doi.org/10.1038%2Fncomms5213.

[27] Jules Tilly, Hongxiang Chen, Shuxiang Cao, Dario Picozzi, Kanav Setia, Ying Li, Edward Grant, Leonard Wossnig, Ivan Rungger, George H. Booth, and Jonathan Tennyson. The variational quantum eigensolver: A review of methods and best practices. *Physics Reports*, 986:1–128, nov 2022. Available from https://doi.org/10.1016%2Fj.physrep.2022.08.003.

[28] Stuart Hadfield. On the representation of boolean and real functions as hamiltonians for quantum computing. *ACM Transactions on Quantum Computing*, 2(4):1–21, dec 2021. Available from https://doi.org/10.1145%2F3478519.

[29] Guillaume Verdon, Michael Broughton, and Jacob Biamonte. A quantum algorithm to train neural networks using low-depth circuits, 2017. Available from https://arxiv.org/abs/1712.05304.

[30] Mark Fingerhuth, Tomáš Babej, and Christopher Ing. A quantum alternating operator ansatz with hard and soft constraints for lattice protein folding, 2018. Available from https://arxiv.org/abs/1810.13411.

[31] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), nov 2018. Available from https://doi.org/10.1038%2Fs41467-018-07090-4.

[32] Zoë Holmes, Kunal Sharma, M. Cerezo, and Patrick J. Coles. Connecting ansatz expressibility to gradient magnitudes and barren plateaus. *PRX Quantum*, 3(1), jan 2022. Available from https://doi.org/10.1103%2Fprxquantum.3.010313.

[33] Earl Campbell, Ankur Khurana, and Ashley Montanaro. Applying quantum algorithms to constraint satisfaction problems. *Quantum*, 3:167, jul 2019. Available from https://doi.org/10.22331%2Fq-2019-07-18-167.

[34] Milan Merkle. Jensen's inequality for medians. *Statistics & Probability Letters*, 71(3):277–281, 2005. Available from https://www.sciencedirect.com/science/article/pii/S0167715204003104.

[35] Bingzhi Zhang, Akira Sone, and Quntao Zhuang. Quantum computational phase transition in combinatorial problems. *npj Quantum Information*, 8(1), jul 2022. Available from https://doi.org/10.1038%2Fs41534-022-00596-2.

[36] Shuichi Sakai, Mitsunori Togasaki, and Koichi Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2):313–322, 2003. Available from https://www.sciencedirect.com/science/article/pii/S0166218X02002056.

[37] R.A. Stoneback, J.H. Klenzing, A.G. Burrell, C. Spence, M. Depew, N. Hargrave, J. Smith, V. von Bose, A. Pembroke, G. Iyer, and S. Luis. Python satellite data analysis toolkit (pysat) vx.y.z, 2021. Available from https://doi.org/10.5281/zenodo.1199703.

[38] John Bent. *Data-Driven Batch Scheduling*. PhD thesis, University of Wisconsin, Madison, May 2005. Available from https://research.cs.wisc.edu/wind/Publications/thesis_johnbent.pdf.

[39] Danny Nam, Allan Sly, and Youngtak Sohn. One-step replica symmetry breaking of random regular nae-sat i, 2021. Available form https://arxiv.org/abs/2011.14270.

[40] Shaowei Cai and Kaile Su. Comprehensive score: Towards efficient local search for sat with long clauses. In *International Joint Conference on Artificial Intelligence*, 2013. Available from https://www.ijcai.org/Proceedings/13/Papers/080.pdf.

[41] Wenbo Sun, Sathwik Bharadwaj, Li-Ping Yang, Yu-Ling Hsueh, Yifan Wang, Dan Jiao, Rajib Rahman, and Zubin Jacob. Limits to quantum gate fidelity from near-field thermal and vacuum fluctuations, 2023. Available from https://arxiv.org/abs/2207.09441.

[42] Xin-Chuan Wu, Sheng Di, Emma Maitreyee Dasgupta, Franck Cappello, Hal Finkel, Yuri Alexeev, and Frederic T. Chong. Full-state quantum circuit simulation by using data compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, nov 2019. Available from https://doi.org/10.1145%2F3295500.3356155.