The Quantum Approximate Optimization Algorithm

Performance on Max-Cut using Heuristic Parameter determination

J.C.P. Bus

TU Delft

The Quantum Approximate Optimization Algorithm

Performance on Max-Cut using Heuristic Parameter determination

by

J.C.P. Bus

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended on Thursday, 23 July 2020 at 10:00.

Student number:	4553896	
Project duration:	January, 2020 – July, 2020	
Thesis committee:	Dr. C. García Almudever,	TU Delft, supervisor
	Dr. M. Möller,	TU Delft, supervisor
	Dr. ir. L.J.J. van Iersel,	TU Delft
	Dr. ir. M. Veldhorst,	TU Delft
	Dr. M. Wimmer,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

The Quantum Approximate Optimization Algorithm (QAOA) is one of the promising near-term algorithms designed to find approximate solutions for combinatorial optimization problems. The algorithm prepares a parametrized state that is aimed to maximize the expectation value of the objective function of the problem. The circuit for QAOA consists of p layers, and it depends on $2p$ parameters, the determination of which can be carried out using a variational quantum eigensolver (VQE) subroutine. This is a variational approach for finding optimal parameters, where the quantum state is prepared with a quantum computer, and a classical computer is used for calculating the corresponding objective function value and performing an outer loop optimization routine in order to find optimal parameters. Earlier work on the algorithm has shown that QAOA is closely related to the Quantum Adiabatic Algorithm, which results in noticeable patterns in the optimal parameters. To this purpose, two methods, INTERP and FOURIER, were proposed to exploit this structure by Zhou et al. [63]. In this thesis I extend upon these results by examining the effectiveness of one of their methods, the INTERP method, on the Max-Cut problem from graph theory. The performance of QAOA using this method was studied on several classes of graphs and benchmarked against the currently best-known classical algorithm Goemans-Williamson, which achieves an approximation ratio of $\rho \approx 0.878$ on arbitrary graphs. The classes of graphs examined include cyclic, weighted and unweighted 3-regular, and Erdős-Rényi graphs with edge probabilities of 0.5 and 0.75 and graph sizes ranging from 4 to 16. Moreover, it was investigated if the INTERP method offers an advantage compared to random initialization of parameters and whether or not the method is polynomial in p. It was found that different graph classes lead to similar but different parameter patterns. Furthermore, the performance of QAOA using the INTERP method was dependent on the graph class considered. For the small graphs investigated it was found that QAOA outperforms Goemans-Williamson on ER-0.50, ER-0.75 and weighted 3-regular graphs from $p = 7$ and beyond with respect to how well the expectation of the objective function approximates the optimal value, while for unweighted 3-regular graphs $p = 6$ suffices. However, the calculation of the parameters is costly and requires a number of function evaluations that is on the order of the number of possible bipartitions of small graph with sizes $n \lesssim 16$. Moreover, the numerical results suggested that indeed the method is polynomial in p advocating that the algorithm might be a strong alternative to Goemans-Williamson for large graph sizes, once hardware allows it.

Contents

1 Introduction

Quantum computing is a radically different kind of computing that makes use of the quantum mechanical nature of matter in order to process information. The two prominent applications of quantum computing you will often read or hear about are Shor's 1994 algorithm [49] and Grover's 1996 algorithm [24], for prime factorization and search problems respectively. However, in order to actually run these algorithms a relatively large number qubits is required, on top of that, fault tolerant quantum computing is needed, requiring even more qubits. As we are now entering the era of Noisy Intermediate-Scale Quantum (NISQ) technology, we are unable to achieve these requirements at the moment. For this reason, we are interested in finding applications that *are* viable to execute on the devices that are presently available, or will be in the near future [47]. In particular, we want to know if there are useful algorithms that can be run on the quantum computers that are available in the short term. Hopefully, by exploring the possibilities, we will find new quantum algorithms that exhibit quantum advantage or even quantum supremacy [3], meaning they offer a speed-up or have capabilities beyond what is tractable on classical computers respectively.

One of the promising algorithms that can be implemented on NISQ devices is the Quantum Approximate Optimization Algorithm, abbreviated QAOA, that was proposed by Farhi, Goldstone and Gutmann in 2014 [20]. The QAOA is an algorithm designed to find approximate solutions for combinatorial optimization problems. It is a hybrid quantum algorithm that makes use of both a classical processor as well as a quantum processor. This approach can be advantageous, especially in NISQ devices, as the need for long coherence times is avoided by only executing short calculations on a quantum computer.

The algorithm prepares a parametrized state after which the state is sampled to infer an approximate solution for the problem. An obstacle for feasible use of the algorithm is the need to find good parameters. Another topic of interest is the performance of QAOA beyond its lowest depth variant, as it is largely unknown.

In this thesis I will explore how QAOA applied to the NP-complete Max-Cut problem compares to currently known classical algorithms, in particular the Goemans-Williamson algorithm [22], the best known classical approximation algorithm. This will be done by using a heuristic technique proposed in [63], called INTERP, for finding suitable, quasi-optimal parameters efficiently. In this work I will expand upon the results of [63] by analyzing the performance of INTERP on different classes of graphs, namely cyclic graphs, 3-regular weighted and unweighted graphs, and Erdős-Rényi graphs with edge probability 0.5 and 0.75. Moreover, the time complexity of the algorithm will be investigated using numerical results of the parameter optimization runs.

Throughout this work I will assume basic knowledge of linear algebra and that the reader is comfortable with terminology from quantum mechanics and quantum computing, such as qubit, superposition, interference and entanglement. For the interested reader without the necessary prerequisites I recommend the canonical *Quantum Computation and Quantum Information* by Michael A. Nielsen and Isaac L. Chuang [44]. For a more gentle and accessible introduction I can highly recommend the website *Quantum Country* [39] by Nielsen and Andy Matuschak. Other good resources I enjoyed reading are [28] and [5].

My thesis will be arranged as follows. I will start with a discussion of combinatorial optimization in Chapter 2, with emphasis on the Max-Cut problem. This will be followed by an overview of QAOA, and previous work done on the topic in Chapter 3. In Chapter 4 I will present a detailed discussion of the implementation of QAOA proposed in [63], which is used to produce the results presented in Chapter 5. I will finalize this thesis with my conclusions and recommendations for future work in Chapter 6.

2

Combinatorial Optimization and Max-Cut

Combinatorial optimization is about finding the optimum (either a maximum or minimum) of an objective function whose domain is discrete, but often very large. Examples of problems considered are the travelling salesman problem [31] important in routing, and the knapsack problem [15, 38] that is a generalization of the struggle you face when you pack your bags for your holiday. Many other problems from day-to-day life, industry and science can be formulated as combinatorial optimization problems. However, exhaustive searches of the domain are ordinarily not tractable. For example, when packing your bag with N items, you have an exponential number of combinations to choose from, namely 2^N. Hence we need smart approaches if we want to tackle these problems. Sadly, even our smartest approaches are sometimes not enough to find optimal solutions efficiently. Instead, we can use approximation algorithms that find good solutions instead. These are algorithms that find for solutions that are as least as good as the optimum value times a certain factor, called the approximation ratio. Sometimes a good solution is all we can ask for.

In this chapter I will show some relevant problems encountered in combinatorial optimization, and explain the need for approximation algorithms. Furthermore, I will explain the Max-Cut problem in more depth together with its applications. Next, I will discuss some classical algorithms for solving it, in particular the state-of-the-art Goemans-Williamson algorithm. Lastly, I will discuss bounds on approximation ratios.

2.1. Combinatorial Optimization Problems

2.1.1. Max-Sat

The Maximum Satisfiability, or Max-Sat, problem is the problem of finding the maximum amount of clauses of a Boolean formula that can be satisfied. It generalizes the Boolean satisfiability problem which asks whether or not there exists an assignment that satisfies all clauses. For example, consider the following Boolean formula

$$\underbrace{(x_0 \vee x_1)}_{\text{clause 1}} \wedge \underbrace{(\neg x_0)}_{\text{clause 2}} \wedge \underbrace{(\neg x_1)}_{\text{clause 3}} \tag{2.1}$$

Observe that no assignment of x_0, x_1 can satisfy all three clauses. There are however assignments that satisfy two clauses, which is optimal. A solution for this particular instance of Max-Sat would be $x_0 = x_1 = 0$. Many other combinatorial optimization problems are closely related to Max-Sat, and can be represented by finding the maximum amount of clauses satisfied of a Boolean formula. One of such related problems is the maximum cut, or Max-Cut problem.

2.1.2. Max-Cut

Given an undirected graph $G = (V, E)$ and non-negative weights $w_{i,j} = w_{j,i}$ on the edges $\{i, j\} \in E$, the maximum cut problem (Max-Cut) is that of finding the set of vertices S that maximizes the sum of the weights of the edges in the cut. I use the term cut to refer to the set of edges with one endpoint in S and the other in $\bar{S} = V \setminus S$, and I will use the terms nodes and vertices interchangeably. The Max-Cut problem is one of Karp's original NP-complete problems [32], and has long been known to be NP-complete even in the case of unweighted graphs. For notational simplicity, I will set $w_{i,j} = w_{j,i} = 0$ if $\{i, j\} \notin E$.

We want to bipartition V into two sets $S \subseteq V$ and $\bar{S} = V \setminus S \subseteq V$ such that the cost function C is maximized

$$C(S) = \sum_{i \in S, j \in \bar{S}} w_{i,j} \tag{2.2}$$

Note that only the edges crossing from S to $\bar{S}$ count towards the objective function as $w_{i,j} = 0$ if $\{i, j\} \notin E$. To relate this to satisfying clauses in the Max-Sat problem from Section 2.1.1, in the context of Max-Cut every edge in the graph is a clause, which is satisfied if the two corresponding vertices are in seperate parts of the bipartition.

For the rest of this thesis, I will denote $V = \{1, \dots n\}$ for simplicity, where n is the number of nodes $n = |V|$. In the most general case $w_{i,j} \in \mathbb{R}$, but sometimes people implicitly take $w_{i,j} = 1$ for all edges $\{i, j\} \in E$, I will refer to these graphs as unweighted graphs. Note that the bipartition is completely characterised by the set $S \subseteq V$. Moreover, note that $C(S) = C(\bar{S})$. For a more intuitive visual representation, please have a look at Figure 2.1.

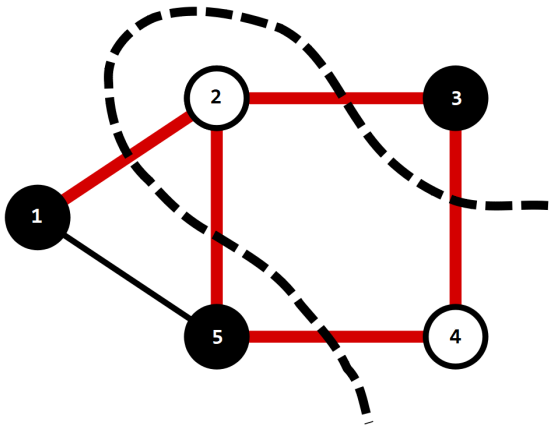


Figure 2.1: An example of the Max-Cut problem on a graph with 5 nodes. The nodes are partitioned into two sets; black nodes and white nodes. The goal is to maximize the number of edges between the two sets (these are colored red). More generally when talking about weighted graphs we want to maximize the sum of the weights of the edges connecting the two sets. Figure from [41]

For reasons that will become apparent in Chapter 3, it is sometimes useful to write the bipartition in terms of a bit string. A bit string is simply a sequence of bits, usually denoted by 0 or 1. This can be achieved by labelling the n vertices in a particular order $(1, \dots, n)$. By using the fact that the partition is completely characterized by *one* set S, we can represent a partition of V into S and $\bar{S}$ by a bit string $\boldsymbol{x}$ consisting of ones and zeros. We can translate between the two using the function $f : \mathcal{P}(V) \mapsto \{0,1\}^n$ defined by

$$f(S) = \big(\mathbf{1}_S(1), \dots, \mathbf{1}_S(n)\big) \tag{2.3}$$

where $\mathcal{P}(V)$ denotes the powerset of V, and $\mathbf{1}_S$ denotes the characteristic function of S. For example, let us label the vertices in Figure 2.1 in clockwise fashion starting from the leftmost vertex. We can represent the cut shown as $\boldsymbol{x} = (1, 0, 1, 0, 1)$, where the black nodes constitute set S and the white nodes consitute set $\bar{S}$. The objective function in terms of $\boldsymbol{x} \in \{0,1\}^n$ becomes

$$C(\boldsymbol{x}) = \sum_{\{i,j\} \in E} w_{i,j}(x_i - x_j)^2 \tag{2.4}$$

Equivalently, we can use a binary string $\boldsymbol{z} \in \{-1, 1\}^n$. The objective function is then described by

$$C(\boldsymbol{z}) = \sum_{\{i,j\} \in E} \frac{w_{i,j}}{2}(1 - z_i z_j) \tag{2.5}$$

This approach will become useful in the next chapter, where we will talk about spins.

2.2. Approximation algorithms and ratios

The Max-Cut problem is an example of a problem that is NP-complete [32]. NP-completeness implies NP-hardness which roughly means we have to resort to extensive searching in order to find the optimal cut. To be more precise, no polynomial-time algorithms for Max-Cut are known for arbitrary graphs. Therefore, we have to resort to finding good solutions instead. One approach to find such good solutions is to design approximation algorithms. These algorithms seek to find solutions that guarantee the objective value to be at least the optimum times a certain ratio, called the approximation ratio ρ.

$$\frac{C(\boldsymbol{x})}{C_{\max}} \geq \rho \tag{2.6}$$

where $C_{\max} = \max_{\boldsymbol{x}} C(\boldsymbol{x})$. Note that $\rho \leq 1$ as we can not do better than the optimal solution. Note that these algorithms grant a guarantee that the objective value for a found solution $\boldsymbol{x}$ lies in a particular range

$$\rho C_{\max} \leq C(\boldsymbol{x}) \leq C_{\max} \tag{2.7}$$

It certainly might be the case that for particular instances of the problem an approximation algorithm finds solutions that are better than $\rho C_{\max}$.

2.2.1. Bounds on the approximation ratio of Max-Cut

In 2002 the Swedish theoretical computer scientist Håstad proved that it is NP-hard to approximate the Max-Cut value with an approximation ratio better than $\frac{16}{17} \approx 0.941$ for Max-Cut on all graphs [30], or $331/333 \approx 0.994$ when restricted to unweighted 3-regular graphs [7]. As the approximation ratio for polynomial-time algorithms for Max-Cut is bounded, this means that the problem is in the class APX, for approximable problems.

In addition, there is a conjecture from theoretical computer science called the Unique Games Conjecture (UGC) [34]. The conjecture postulates that the problem of determining the approximate value of a unique game, is NP-hard. The details of the Unique Games Conjecture are rather technical and not relevant for the rest of this work, I refer the interested reader to the original paper [34], or to [36] for a less formal discussion. However, there is a consequence of the conjecture that is relevant. It turns out that if the UGC is true, then there exists no polynomial time algorithm that can beat Goemans-Williamson [35], thus restricting the possible approximation ratio even further to around 0.878.

At the moment, it is unknown whether or not the conjecture is true, but it is interesting to note that an approximation algorithm that is better than Goemans-Williamson would imply that the UGC is false. Unlike the question whether or not $P = NP$, currently there seems to be no consensus in the academic world around the Unique Games Conjecture.

2.3. Classical Algorithms for Max-Cut

In this section I will discuss known classical algorithms for solving Max-Cut to give a sense of what is already possible on classical computers. The presentation of the algorithms in the coming subsections largely follows the structure of [10, 11].

2.3.1. Greedy approach

One algorithm for solving Max-Cut uses a greedy approach and guarantees an approximation ratio of $\frac{1}{2}$.

Let $G(V,E)$ be an undirected graph with edge weights w_e for $e \in E$ and label $V = \{1,\dots n\}$ so there is some ordering. Next, define $E_i = \{(k,i) : \{k,i\} \in E, k < i\}$, the set of edges entering vertex i from "lower" nodes. Note that $E_i \cap E_j = \emptyset$ if $i \neq j$ because of the strict inequality. Initially, set $S = \{1\}$ and $\bar{S} = \emptyset$. Then starting from $i = 2$, add node i to either S or $\bar{S}$ so that the sum of the number of edges in E_i that are in the cut times their respective weights is maximized. Increment i and repeat the last step until $i = n$.

As an example, consider the graph from Figure 2.1. For simplicity I will consider it to be an unweighted graph with $w_{i,j} = 1$ for all $\{i,j\} \in E$. Starting with $S = \{1\}$, consider node 2. We choose to add node 2 based on what choice maximizes $\sum_{e \in E_2 \cap K(S)} w_e$, which in our case is simple: we add node 2 to set $\bar{S}$ because then edge $(1,2)$ is in the cut. We move on to node 3 with $E_3 = \{(2,3)\}$, and we choose node 3 to be in S because then edge $(2,3)$ is in the cut. Analogously, we add edge 4 to $\bar{S}$. Node 5 is somewhat more tricky as we have $E_5 = \{(1,5),(2,5),(4,5)\}$ containing 3 edges instead of just one. Adding node 5 to S would yield $\sum_{e \in E_5 \cap K(S)} w_e = 2$ while adding it to $\bar{S}$ would yield $\sum_{e \in E_5 \cap K(S)} w_e = 1$. Therefore, we add node 5 to S and we

are done. The final result is thus $S = \{1,3,5\}$ and $\bar{S} = \{2,4\}$ with a cutvalue of 5, which is actually optimal.

Let us proof that this is indeed an approximation algorithm with approximation ratio $\frac{1}{2}$. Define K to be the cut obtained by the algorithm, and observe that $E_1, \dots E_n$ is a partition of E as the sets are pairwise disjoint and $E = \bigcup_i E_i$. For each node $i \in V$, define $\tilde{E}_i = E_i \cap K$. Note that the cut K is the union of these newly constructed sets $K = \bigcup_i \tilde{E}_i$. The finalizing observation is that we chose $S, \bar{S}$ in such a way that

$$\sum_{e \in \tilde{E}_i} w_e \geq \frac{1}{2} \sum_{e \in E_i} w_e \qquad \forall i \in V \tag{2.8}$$

Therefore we have that

$$C(S) = \sum_{i \in V} \sum_{e \in \tilde{E}_i} w_e \geq \frac{1}{2} \sum_{i \in V} \sum_{e \in E_i} w_e = \frac{1}{2} \sum_{e \in E} w_e \geq \frac{1}{2} C_{\max} \tag{2.9}$$

where the latter inequality is derived from the fact that the optimum cut can not be larger than the sum of the weights of all edges.

2.3.2. Random partition

Another algorithm for solving Max-Cut that is quite easy to implement is simply choosing a random partition. As the name suggests, it is a stochastic algorithm.

Let $G = (V, E)$ again be an undirected graph. We choose each vertex in V to be in either S or $\bar{S}$ with equal probability $\frac{1}{2}$. This bipartition defines a cut $K(S)$, whose expected size we will now calculate.

Consider an edge $e \in E$ in the graph. Remark that $\Pr[e \in K(S)] = \frac{1}{2}$. For $e \in E$, we define χ_e to be a random variable such that $\chi_e = 1$ if $e \in K(S)$ and $\chi_e = 0$ if $e \notin K(S)$. The value of the cut is then

$$C(S) = \sum_{e \in E} w_e \chi_e \tag{2.10}$$

Note that the value itself is now a random variable, so we have to resort to calculating the expecation value of the cut. Using linearity of the expectation operator, we obtain the following

$$\mathrm{E}[C(S)] = \sum_{e \in E} w_e \mathrm{E}[\chi_e] = \sum_{e \in E} w_e \Pr[e \in K(S)] = \frac{1}{2} \sum_{e \in E} w_e \geq \frac{1}{2} C_{\max} \tag{2.11}$$

Resulting in a $\frac{1}{2}$-approximation ratio in expectation. I would like to emphasize that we give a lower bound for the expectation value of the objective function, and not a lower bound for every outcome. Note that it might be the case that the algorithm produces with probability 2^{-n} a partition $S = V, \bar{S} = \emptyset$ with a cutvalue of 0.

2.3.3. Goemans-Williamson

The best classical approximation algorithm for solving Max-Cut known is the algorithm proposed by Goemans and Williamson (GW) in 1994 [22]. The algorithm is also stochastic and has an approximation ratio $\rho \approx 0.878$ in expectation. Prior to GW, the best-known approximation algorithms for the Max-Cut problem had performance guarantee of $\frac{1}{2} + o(1)$, meaning that the achieved approximation ratio is below $\frac{1}{2} + \epsilon$ for any $\epsilon > 0$. The GW-algorithm was indeed quite a breakthrough.

The algorithm uses a sophisticated method that randomly rounds the solution to a nonlinear programming relaxation. Before I will present the algorithm, the following two definitions are useful.

Definition 2.3.1. A square $n \times n$ matrix A is called **positive semidefinite** if the following holds

$$\boldsymbol{x}^T A \boldsymbol{x} \geq 0 \qquad \forall \boldsymbol{x} \in \mathbb{R}^n \tag{2.12}$$

This property is sometimes indicated using $A \succcurlyeq 0$

Definition 2.3.2. A **semidefinite program** is the problem of optimizing a linear function of a symmetric positive semidefinite matrix subject to linear equality constraints.

Indeed a semidefinite program is a subclass of the more general linear programs, so it is possible to solve these using the well known Simplex method [14]. While the Simplex method itself is not a polynomial time algorithm, it can be generalized to semidefinite programs [45]. Given any $\epsilon > 0$ these semidefinite programs can be solved in polynomial time within an additive error of ϵ. This can be done using a variety of methods,

these include interior point methods [43], the ellipsoid method [37], and other polynomial-time algorithms for convex programming [53].

The problem to be solved is

$$\underset{\boldsymbol{z}}{\text{maximize}} \quad \frac{1}{4}\sum_{\{i,j\}\in E} w_{i,j}(z_i - z_j)^2 \tag{2.13a}$$

$$\text{subject to} \quad z_i \in \{-1,1\} \text{ for } i \in V \tag{2.13b}$$

which is equivalent to what we saw in (2.4). Let us denote the optimal solution of this program as $C_{\max}$. Next, we introduce a *relaxation* of the problem meaning that we let loose of some of the constraints in order to make solving it easier

$$\underset{\boldsymbol{z}}{\text{maximize}} \quad \frac{1}{4}\sum_{\{i,j\}\in E} w_{i,j}(\boldsymbol{y}_i \cdot \boldsymbol{y}_i + \boldsymbol{y}_j \cdot \boldsymbol{y}_j - 2\boldsymbol{y}_i \cdot \boldsymbol{y}_j) \tag{2.14a}$$

$$\text{subject to} \quad \boldsymbol{y}_i \in \mathbb{R}^n \text{ and } \|\boldsymbol{y}_i\|^2 = 1 \text{ for } i \in V \tag{2.14b}$$

Note that the above problem is indeed a relaxation, as we can simply take $\boldsymbol{y}_i = z_i \boldsymbol{e}_1$ to reconstruct program (2.13a), (2.13b), where $\boldsymbol{e}_1$ is the first standard basis vector of $\mathbb{R}^n$. Let me denote the optimal solution of the relaxation as $\tilde{C}_{\max}$, then we have $\tilde{C}_{\max} \geq C_{\max}$. The above so called vector program is equivalent to the following semidefinite program, where we define a matrix $A = (a_{ij})$

$$\underset{\boldsymbol{z}}{\text{maximize}} \quad \frac{1}{4}\sum_{\{i,j\}\in E} w_{i,j}(a_{ii} + a_{jj} - 2a_{ij}) \tag{2.15a}$$

$$\text{subject to} \quad a_{ii} = 1 \text{ for } i \in V \text{ and } A \succcurlyeq 0 \text{ and symmetric} \tag{2.15b}$$

This semidefinite program can be solved in polynomial time obtaining an optimal matrix A^* with value c. Note that the solution could be irrational. In that case we can find an approximation that is rational in polynomial time with value c' with $c - \epsilon < c' < c$ for any $\epsilon > 0$. Using the symmetry of the semidefinite matrix A^* we know that we can decompose $A^* = U^T U$ where U is an $m \times n$ matrix with $m \leq n$, this is called the Cholesky decomposition. Next, we consider the column vectors of $U = \{\boldsymbol{u}_1, \dots \boldsymbol{u}_n\}$. Because of the semideniteness of A^* and the constraint that $a^*_{ii} = 1$, we know that these are unit vectors $\|\boldsymbol{u}_i\|^2 = \boldsymbol{u}_i^T \boldsymbol{u}_i = a^*_{ii} = 1$. Moreover, we know that $\boldsymbol{u}_i \in \mathbb{R}^m$ for $i = 1, \dots, n$ and since $m \leq n$ we can embed these vectors in $\mathbb{R}^n$. Thereafter, we randomly choose a hyperplane through the origin, and in polynomial time we can figure out which vectors are above or below the hyperplane. The vectors $\boldsymbol{u}_i$ that lay above the plane denote vertices that are on one side of the partition, and the vectors $\boldsymbol{u}_i$ below the plane denote the vertices that are on the other side of the partition.

Using a combination of geometric arguments and insights from calculus, it can be shown that the algorithm has an approximation ratio of

$$\rho = \frac{2}{\pi} \min_{0 \leq \theta \leq \pi} \frac{\theta}{1 - \cos\theta} \tag{2.16}$$

in expectation, which numerically evaluates to $\rho \approx 0.878$ [22].

For a more extensive discussion of the Goemans-Williamson algorithm I refer to [10, 22, 29].

2.4. Applications of Max-Cut

The maximum cut problem is a natural and simple to understand graph problem often discussed in introductory optimization or algorithm courses. There are also some applications of the problem. An example of which is efficient design of electric circuits or communication networks [57], but the problem also has uses in statistical physics, namely in finding the groundstate energy in spin glass [6].

2.4.1. Ising Model

The Ising model describes a system of n particles with two possible states, for example spin up and down. See Figure 2.2 for an illustration. The system can then be characterized by a binary string $\sigma \in \{-1,1\}^n$. Following the example, 1 denotes spin up and -1 denotes spin down. The Hamiltonian reads as follows

$$H(\sigma) = \sum_{\{i,j\}} J_{ij}\sigma_i\sigma_j - \mu\sum_i h_i\sigma_i \tag{2.17}$$

where the first sum is over the adjacent pairs $\{i, j\}$ in the system, and J_{ij} designates the strength of the interaction of the constituents of that pair. The second sum describes the interaction with some magnetic field, with strength h_i at site i. Note that when there is no external magnetic field, i.e. $h_i = 0$ for all i, then the problem of finding a groundstate is equivalent to the Max-Cut problem. Closely related to the Ising model is the Sherrington-Kirkpatrick model.

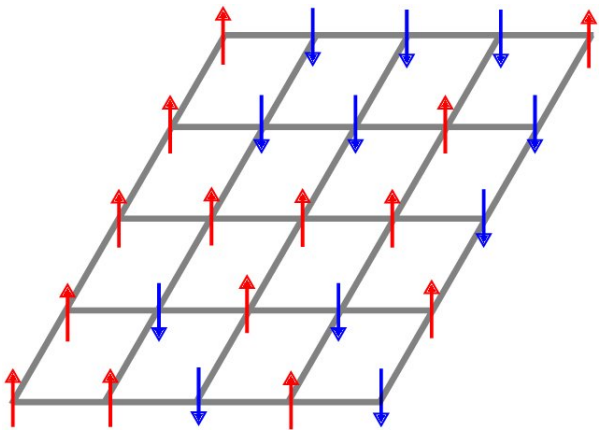


Figure 2.2: Schematic representation of a configuration of the 2D Ising model on a square lattice. Figure and caption from [56]

3

The Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm, abbreviated QAOA, was proposed by Farhi, Goldstone and Gutmann in 2014 [20]. The goal of the algorithm is finding approximate solutions to combinatorial optimization problems. Due to its shallow depth it is feasible to run it without error correction on NISQ devides, making it interesting for near-term implementation. The quantum circuit that implements the algorithm prepares a parametrized state, and consists of p layers. Each of the layers comprises of two unitary operators, the first encoding the problem and the latter entangling the qubits, parametrized by some angles. Appropriate parameters can be found in a variety of ways, the most popular being a variational approach similar to the variational quantum eigensolver [46]. By sampling from this parametrized state an approximate solution to the problem can be found.

There are two main reasons why QAOA is promising for solving combinatorial optimization problems. First of all because it can be shown that adding extra layers monotonically improves the result of the algorithm. Additionally, in the limit $p \to \infty$, the algorithm produces the optimum, similar to the quantum adiabatic algorithm [20]. Secondly, it is interesting because of its potential to exhibit quantum supremacy [17], as it can not be efficiently simulated by a classical computer. Because of this potential, the algorithm has received quite some attention from the scientific community in the last few years and continues to do so.

In this chapter, I will give an overview of prior work that has been done on QAOA. Specifically, I will explain the general algorithm, how the algorithm can be applied to Max-Cut and how it relates to the Quantum Adiabatic Algorithm that was proposed in [18] in 2000.

3.1. General setup

The Quantum Approximate Optimization Algorithm is designed to handle general optimization problems. These problems can be specified by n bits and m clauses, where a clause is a constraint on a subset of the bits which is satisfied for certain assignments of those bits and not satisfied for other assignments. We want to find a bit string $\boldsymbol{x} \in \{0,1\}^n$ of size n that maximizes or minimizes a cost function [20]

$$C(\boldsymbol{x}) := \sum_{\alpha=1}^{m} C_\alpha(\boldsymbol{x}) \tag{3.1}$$

where we sum over m clauses that depend on a subset of the n bits. We define $C_\alpha = 1$ if clause α is satisfied and $C_\alpha = 0$ otherwise. The goal is to satisfy as many clauses as possible, as that we want to maximize $C(\boldsymbol{x})$. The QAOA is a quantum algorithm that is aimed to find a bit string $\boldsymbol{x}'$ for which $C(\boldsymbol{x}')$ is close to the maximum of C. A quantum computer works in a 2^n dimensional vector space, more precisely a Hilbert space in which the norm is induced by an inner product. The computational basis vectors $|\boldsymbol{x}\rangle$ can be represented by bit string consisting of 0s and 1s. We can consider both the objective function C and the clause C_α in (3.1) as operators

that are diagonal in the computational basis.

$$\hat{C} = \sum_{\boldsymbol{x}\in\{0,1\}^n} C(\boldsymbol{x}) \cdot |\boldsymbol{x}\rangle\langle\boldsymbol{x}| = \sum_{\alpha=1}^{m} \hat{C}_\alpha \qquad \text{where} \qquad \hat{C}_\alpha = \sum_{\boldsymbol{x}\in\{0,1\}^n} C_\alpha(\boldsymbol{x}) \cdot |\boldsymbol{x}\rangle\langle\boldsymbol{x}| = \begin{bmatrix} C_\alpha(0,\ldots,0) & & \\ & \ddots & \\ & & C_\alpha(1,\ldots,1) \end{bmatrix} \tag{3.2}$$

We call $\hat{C}$ the **cost Hamiltonian**. To avoid confusion, I will refer to the operators above as $H_C \equiv \hat{C}$ and $H_{C_\alpha} \equiv \hat{C}_\alpha$ for the rest of this thesis, following the notation from [28]. Using the operator H_C, we define the operator $U(H_C,\gamma)$ which depends on some real valued angle $\gamma \in \mathbb{R}$ as follows

$$U(H_C,\gamma) := e^{-i\gamma H_C} = \prod_{\alpha=1}^{m} e^{-i\gamma H_{C_\alpha}} \tag{3.3}$$

I will refer to this object as the **cost unitary**. Note that the cost Hamiltonian from (3.2) is Hermitian, and therefore has purely real eigenvalues, hence the operator $U(H_C,\gamma)$ is indeed unitary. Moreover, observe that the latter equality from (3.3) comes with a caveat. As we want to be able to write $U(H_C,\gamma)$ as a product of local terms it is important that the H_{C_α} commute pairwise.

Remark. Remember that the definition of the matrix exponential is derived from the Taylor expansion of the exponent. The exponential of matrix A is given by

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k = I + A + \frac{1}{2}A^2 + \frac{1}{6}A^3 + \ldots \tag{3.4}$$

Different than taking the exponential of real numbers, the following equality does not always hold for matrix exponents because of possible noncommutativity.

$$e^{A+B} \stackrel{?}{=} e^A e^B \tag{3.5}$$

Observe that

$$e^{A+B} = I + (A+B) + \frac{1}{2}(A+B)^2 + \cdots = I + A + B + \frac{1}{2}(A^2 + AB + BA + B^2) + \ldots \tag{3.6}$$

$$e^A e^B = (I + A + \frac{1}{2}A^2 + \ldots)(I + B + \frac{1}{2}B^2 + \ldots) = I + A + B + \frac{1}{2}(A^2 + 2AB + B^2) + \ldots \tag{3.7}$$

It can be shown that (3.5) only holds if A and B commute, this statement can be extended for arbitrary finite sums. Hence, it is essential for H_{C_α} to commute when separating the cost unitary into local terms in (3.3).

If we give every clause equal weight $w_\alpha = 1$, we know that the cost Hamiltonian is diagonal with solely integer entries, which are precisely the eigenvalues of the matrix. We can use the fact that H_C has integer eigenvalues in order to restrict γ to be in $[0, 2\pi)$. The approach of translating an objective function to an operator is very general and can be applied to a variety of problems. The actual form of H_C is be determined by the specific problem. In section 3.2 I will discuss it in the case of Max-Cut.

Next, we define the operator H_B, called the **mixer Hamiltonian**

$$H_B := \sum_{j=1}^{n} \sigma_x^{(j)} \tag{3.8}$$

where $\sigma_x^{(j)}$ is the Pauli-X gate acting solely on the qubit j.

$$\sigma_x^{(j)} = I^{\otimes j-1} \otimes \sigma_x \otimes I^{\otimes n-j} \tag{3.9}$$

where I is the 2×2 identity matrix, $\otimes$ denotes the Kronecker product and $A^{\otimes n}$ denotes the Kronecker product of a matrix A with itself n times.

$$A^{\otimes n} = \underbrace{A \otimes \cdots \otimes A}_{n \text{ times}} \tag{3.10}$$

Using the mixer Hamiltonian, we can define another unitary operator, the **mixer unitary**, similar to what we did before

$$U(H_B,\beta) := e^{-i\beta H_B} = \prod_{j=1}^{n} e^{-i\beta\sigma_x^{(j)}} \tag{3.11}$$

where again $\beta \in \mathbb{R}$ is some real angle. Note that $\sigma_x^{(j)}$ commute pairwise so when taking the exponential of H_B that is the sum of these terms, expression (3.5) holds and we are able to construct H_B with local gates. With these unitaries defined, we can move on to the next part: preparing the parameterized state. We start of in the initial equal superposition state

$$|s\rangle := \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle = |+\rangle^{\otimes n} = H^{\otimes n}|0\rangle^{\otimes n} \tag{3.12}$$

which is easy to prepare using Hadamard gates applied to each of the n qubits, all in the state $|0\rangle$. Thereafter, we alternately apply the cost Hamiltonian and the mixer Hamiltonian p times, using different parameters $\gamma_1 \dots \gamma_p$ and $\beta_1 \dots \beta_p$, respectively.

$$|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle := \underbrace{U(H_B, \beta_p)U(H_C, \gamma_p)\dots \underbrace{U(H_B, \beta_1)U(H_C, \gamma_1)}_{\text{one layer}}}_{p \text{ layers}} |s\rangle \tag{3.13}$$

I use boldface vector notation $\boldsymbol{\gamma}, \boldsymbol{\beta}$ for the sole purpose of writing these sets of parameters more compactly. Furthermore, take note that we work from right to left, as is convention when working with matrices. See Figure 3.1 for a schematic of the circuit preparing the parametrized state. Now that we have established this, we can move on to the actual aim of this circuit. The idea is to find parameters such that

$$F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) := \langle \boldsymbol{\gamma}, \boldsymbol{\beta} | H_C | \boldsymbol{\gamma}, \boldsymbol{\beta} \rangle \tag{3.14}$$

is maximized, where we indicate the number of layers p using a subscript. In general, $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^p$, however depending on the cost and mixer unitaries we can restrict the domain using symmetries. We define

$$M_p := \max_{\boldsymbol{\gamma}, \boldsymbol{\beta}} F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) \tag{3.15}$$

and want to find the corresponding parameters. By sampling from this state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ we can find an approximate solution by looking at either the most sampled bit string, or the best sampled bit string.

The reader might wonder why we need to apply the mixer unitary when only the cost unitary encodes the problem. This has to do with the way probabilities work in quantum mechanics. The probability of measuring a bit string that encodes a particular solution, is precisely the square magnitude of the corresponding amplitude. Take note of the fact that the exponent of a diagonal $n \times n$ matrix $A = \text{diag}\{a_{11}, \dots a_{nn}\}$ is a diagonal matrix with the entries exponentiated.

$$e^{\text{diag}\{a_{11}, \dots a_{nn}\}} = \text{diag}\{e^{a_{11}}, \dots, e^{a_{nn}}\} \tag{3.16}$$

Since the cost Hamiltonian H_C is diagonal and real, the cost unitary will be diagonal with entries of the form $e^{i\phi}$ for some $\phi \in \mathbb{R}$. This implies that the amplitudes will only acquire relative phases, and thus the probability of finding a particular bit string is not affected by the cost unitary. If we were going to run a circuit with only cost unitaries, it would result in an equal superposition, whatever set of angles $\boldsymbol{\gamma}$ we choose. A fancy way of implementing the random partition algorithm from Section 2.3.2. For this reason we need an operator that entangles the states to do better than the random partition algorithm, hence the need for the mixer Hamiltonian.

Finding the optimal, or even a good set of angles $\boldsymbol{\gamma}, \boldsymbol{\beta}$ is a non-trivial task. In Section 3.3 a more thorough discussion on this topic can be found. Furthermore, one method for finding the angles proposed in [63], called INTERP, is discussed in more depth in Chapter 4, which will also be used to produce the results in Chapter 5.

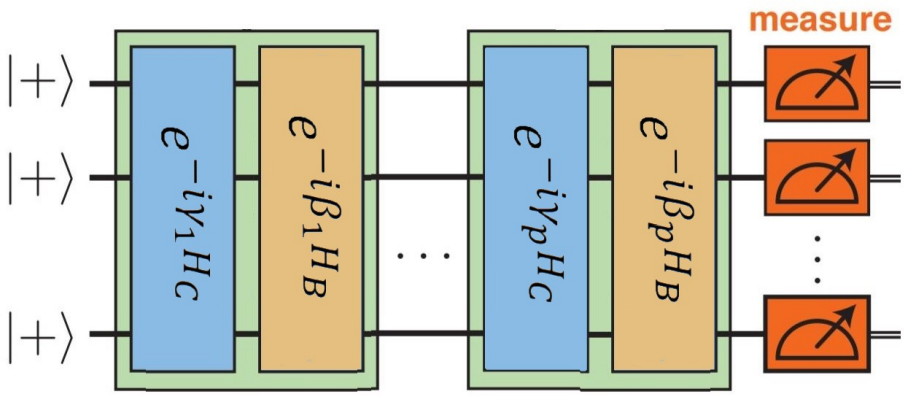


Figure 3.1: Schematic of a p-level Quantum Approximation Optimization Algorithm. The cost unitary and the mixer unitary are alternately applied to prepare the parametrized state $|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle$. Adapted figure from [63].

3.2. Applying QAOA to Max-Cut

QAOA can easily be applied to the Max-Cut problem by associating each vertex $i \in V$ with a qubit q_i and using the qubit's state to encode the affiliation to one of the sets of the bipartition. A qubit in state $|1\rangle$ means it is in one half of the bipartition, S, a qubit in state $|0\rangle$ means it is in the other, namely $\bar{S}$. Our aim is to find a cut such that we maximize the sum of the edges in the cut, so we would like adjacent qubits to be in different halves of the bipartition. As explained in Section 2.1.2, we can express this mathematically as

$$C(\boldsymbol{z}) = \frac{1}{2}\sum_{\{i,j\}\in E} w_{i,j}(1 - z_i z_j) \tag{3.17}$$

where $z_i \in \{-1,1\}$ for $i \in V$. Now the idea is to translate this problem into a cost Hamiltonian. Remark that the Pauli matrix $\sigma_z = \text{diag}\{1,-1\}$ has eigenvalues 1 and -1. Therefore, we can use $\sigma_z^{(i)}$ to represent z_i. Using this we can define the cost Hamiltonian to be

$$H_C = \frac{1}{2}\sum_{\{i,j\}\in E} w_{i,j}\left(I - \sigma_z^{(i)}\sigma_z^{(j)}\right) \tag{3.18}$$

Note that the term $I-\sigma_z^{(i)}\sigma_z^{(j)}$ involving qubits i and j is diagonal in the computational basis. This is important because we want to be able to write the cost unitary as a product of local terms

$$U(H_C,\gamma) = e^{-i\gamma H_C} = \prod_{\alpha=1}^{m} e^{-\gamma H_{C\alpha}} = \prod_{\{i,j\}\in E} e^{-\gamma\frac{w_{i,j}}{2}\left(I-\sigma_z^i\sigma_z^j\right)} \tag{3.19}$$

Since all the terms in the product are diagonal, they all commute as desired. Now we have all the ingredients to prepare the state $|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle$ for Max-Cut using the unitary operators $U(H_C,\gamma)$ and $U(H_B,\beta)$.

The actual outcome of the algorithm is determined by sampling from the state $|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle$ and calculating the cut value corresponding to the measured bit string on a classical computer. This yields a distribution of bit strings (and cutvalues). However, in the end we want one solution and not a distribution. One option would be to pick the most sampled bit string, but performance is easily improved by picking the best sampled bit string instead.

Now, we only need to find out how to implement this in a circuit and figure out a way for finding appropriate angles $\boldsymbol{\gamma},\boldsymbol{\beta}$. This will be discussed in Subsection 3.2.1 and Section 3.3, respectively.

3.2.1. Quantum circuit design

It is useful to know about the rotation operators R_x and R_z when wanting to implement the unitary operators $U(H_C,\gamma), U(H_B,\beta)$. These operators induce rotations around their respective axes, and are defined as follows

$$R_x(\theta) \equiv e^{-\frac{\theta i}{2}\sigma_x} = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \tag{3.20}$$

$$R_z(\theta) \equiv e^{-i\frac{\theta}{2}\sigma_z} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix} \tag{3.21}$$

Furthermore, we need a CNOT gate [44], which involves 2 qubits

$$\text{CNOT}_{q_i,q_j} = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \quad q_i \quad q_j \tag{3.22}$$

To understand this component intuitively, the target qubit q_j will be flipped if the control qubit q_i is 1. As we are working with qubits, this does not encapsulate the complete picture, so we need to describe it with a matrix instead.

The X-interactions in the mixer Hamiltonian can be implemented with this one-qubit gate R_x.

$$e^{-i\beta\sigma_x^{(i)}} \equiv \quad q_i \quad \boxed{R_x(2\beta)} \tag{3.23}$$

The two-qubit ZZ-interactions in the cost unitary can be implemented with two CNOT gates, and the local one-qubit gate R_z [13]. Since $\sigma_z^{(i)}$ and $\sigma_z^{(j)}$ commute the roles of q_i and q_j are interchangable.

$$e^{-\frac{i}{2}\gamma(I-\sigma_z^{(i)}\sigma_z^{(j)})} \equiv \quad q_i \quad q_j \quad \boxed{R_z(-\gamma)} \quad \equiv \quad q_i \quad q_j \quad \boxed{R_z(-\gamma)} \tag{3.24}$$

As both $U(H_C,\gamma)$ and $U(H_B,\beta)$ are both products of local terms, we only need those terms to implement them. The final circuit first applies Hadamards to each qubit in order to create an equal superposition over all bit strings, or possible partitions if you will. Next we apply the cost unitary and the mixer unitary in an alternating fashion, in that order, with desired angles. Lastly, we measure the state to get an actual bit string.

As an example, consider the unweighted *diamond graph*, visualised in Figure 3.2.

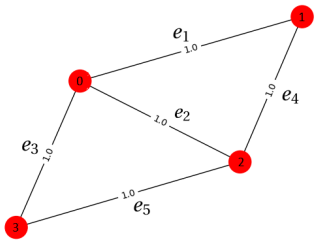


Figure 3.2: Diamond graph (unweighted)

The graph consists of 4 nodes, whereof 2 nodes connect to 2 edges, and 2 nodes connect to 3 edges. The circuit needed to prepare the state $|\gamma,\beta\rangle$ with $p = 1$ for this particular graph is given in Figure 3.3. In general, the depth of the circuit, assuming no overhead necessary for swapping is then at most $p(3m+n)$ where p is the number of QAOA layers, n is the number of nodes, or qubits and m is the number of edges in the graph. For a graph with n nodes we can bound m by $\frac{1}{2}n(n-1)$, the number of edges in a complete graph. Concluding, we see that the circuit requires $O(n^2)$ gates assuming full qubit connectivity.

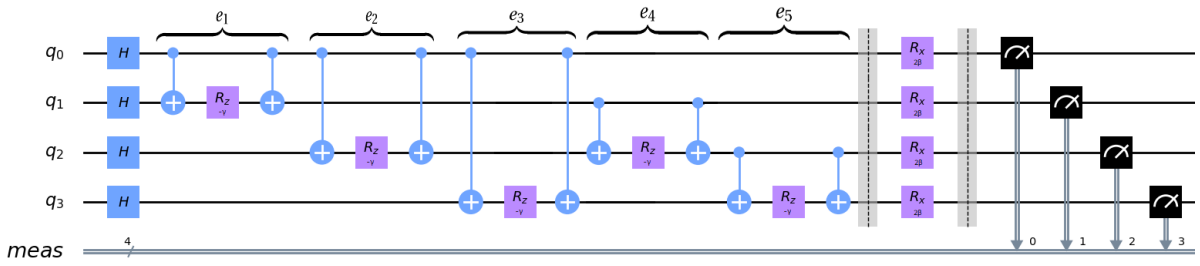
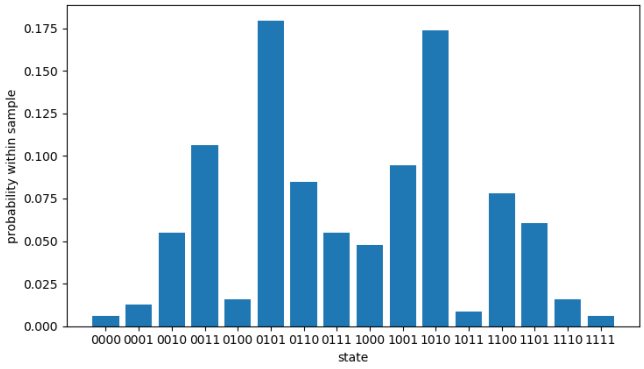


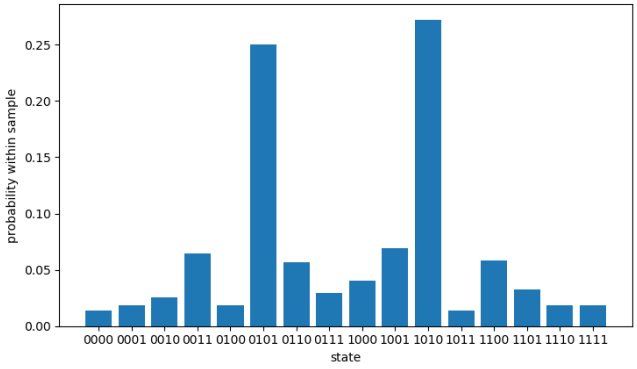
Figure 3.3: Circuit for the diamond graph from Figure 3.2 with parameters γ and β for the rotations using the cost and mixer Hamiltonians respectively. Every node in the graph is represented by a qubit, and the cost unitary can be implemented with two CNOT gates with a $R_z(-\gamma)$ gate in between for every edge in the graph, the relation to the graph edges is indicated with the braces. After the cost unitary, the mixer unitary is applied. This operator can be broken down into rotation gates $R_x(2\beta)$ applied to every qubit, or node. Finally, the system is measured in the computational basis.

3.2.2. Maximizing the expectation value

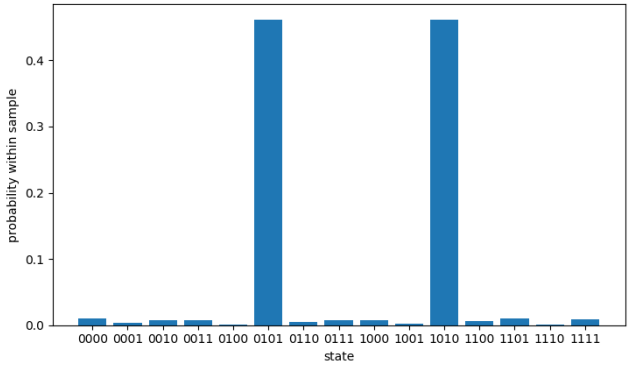
To give a sense of what it means to improve the expectation value of the cost function, several barplots are shown of the distribution resulting from sampling from three different states $|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle_{(p)}$ for $p = 1,2,3$ after optimizing the angles for the diamond graph. As we can see, as we increase p we are able to improve the expecation value and get good bit strings with high probability. In Section 3.3 I will go into more depth on how to determine an appropriate set of angles.



(a) $p = 1$



(b) $p = 2$



(c) $p = 3$

Figure 3.4: Histograms of the distribution when sampling from the state $|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle_{(p)}$ for $p = 1,2,3$ with $\boldsymbol{\gamma},\boldsymbol{\beta} \in \mathbb{R}^p$. The three sets of parameters are determined using BFGS optimization starting with a randomly chosen intial point. For these angles we have $F_1 = 3.24$, $F_2 = 3.39$, $F_3 = 3.87$. The order of the bits is $(0,1,2,3)$ where the labelling of the nodes is the same as in Figure 3.2. The optimal partitions are 0101 and 1010 with cutvalue 4.

3.2.3. Expectation landscapes

To provide insight to what kind of expectation landscapes we might encounter, I will discuss the relevant features and symmetries that come into play in this section. In addition, I will show some expectation landscapes in the case of $p = 1$.

For unweigthed graphs we find periodicity in the γ direction. As mentioned in Section 3.1, this has to do with the fact that the cost Hamiltonian H_C has integer eigenvalues, resulting in the identity

$$e^{2\pi i H_C} = I \tag{3.25}$$

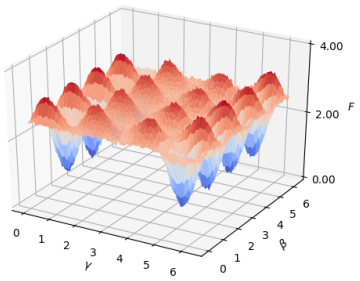
yielding a period of 2π in the γ parameter in all layers. Moreover, there is $\mathbb{Z}_2$ symmetry to be exploited. This means that the partition $(S, \bar{S})$ is equivalent to the partition $(\bar{S}, S)$, or in terms of bit strings a partition 10101 is equivalent to 01010 as the inversed labelling does not alter the value of a cut. Therefore if we apply $X^{\otimes n}$ to the state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ it does not change the expectation value $F_p = \langle H_C \rangle$. Note that the mixer unitary can be constructed with $R_x(2\beta)$ gates, but since $R_x(2\pi) \hat{=} X$ (up to global phase) we find a periodicity of π in the β parameter. Hence we can restrict $\beta \in [0, \pi)$ in general. Concluding, for unweighted graphs we have

$$\langle \boldsymbol{\gamma}, \boldsymbol{\beta} | H_C | \boldsymbol{\gamma}, \boldsymbol{\beta} \rangle = \langle \boldsymbol{\gamma} + 2\pi \boldsymbol{n}, \boldsymbol{\beta} + \pi \boldsymbol{m} | H_C | \boldsymbol{\gamma} + 2\pi \boldsymbol{n}, \boldsymbol{\beta} + \pi \boldsymbol{m} \rangle \qquad \boldsymbol{n}, \boldsymbol{m} \in \mathbb{N}^p \tag{3.26}$$

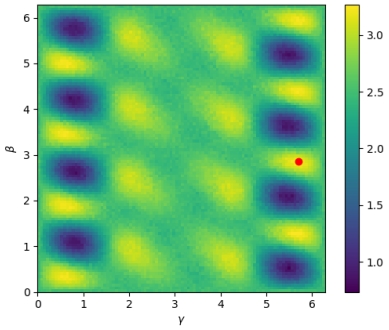
and for weighted graphs we have

$$\langle \boldsymbol{\gamma}, \boldsymbol{\beta} | H_C | \boldsymbol{\gamma}, \boldsymbol{\beta} \rangle = \langle \boldsymbol{\gamma}, \boldsymbol{\beta} + \pi \boldsymbol{m} | H_C | \boldsymbol{\gamma}, \boldsymbol{\beta} + \pi \boldsymbol{m} \rangle \qquad \boldsymbol{m} \in \mathbb{N}^p \tag{3.27}$$

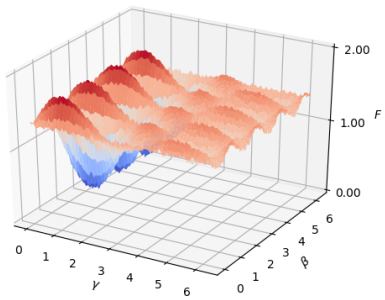
In Figures 3.5 and 3.6 some examples of F_1 landscapes are shown. Remember that $F_1 = \langle \gamma, \beta | H_C | \gamma, \beta \rangle$ was defined to be the expectation value of the cost Hamiltonian. This means that if we repeatedly prepare the state $|\gamma, \beta\rangle$ and calculate the cut values, the average will be $F_1(\gamma, \beta)$. As explained, the aim is to seek values for γ, β that maximize this quantity.



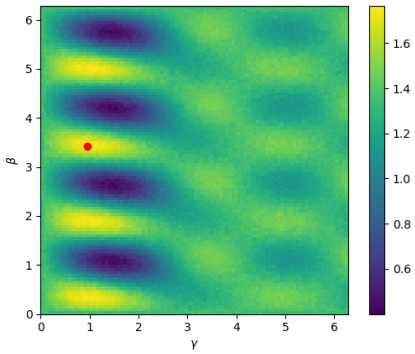
(a) 3D plot of F_1 for the unweighted diamond graph



(b) Colorplot of F_1 for the unweighted diamond graph

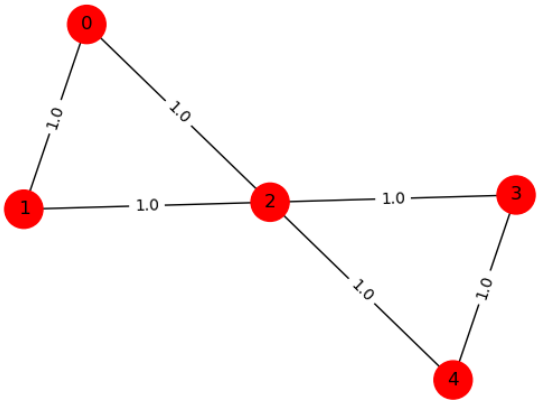


(c) 3D plot of F_1 for the weighted diamond graph.

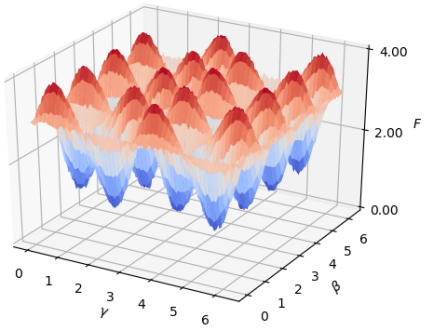


(d) Colorplot of F_1 for the weighted diamond graph

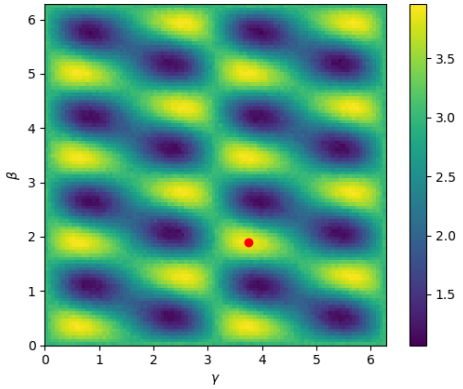
Figure 3.5: Landscapes of the expectation value F_1 for the unweighted (a), (b) and weighted (c), (d) Diamond graph (Figure 3.2) using a 3D plot and a colorplot respectively. The weights for the weighted graph were drawn from a uniform distribution $w_{ij} \sim U(0,1)$ for the weights of edges $\{i, j\} \in E$. In all plots both γ, β range from 0 to 2π, divided into 100 gridpoints. The expectation value $F_1 = \langle H_C \rangle$ is estimated using 1024 samples from the state $|\gamma, \beta\rangle$ at each gridpoint. The maximum is indicated by a red dot in the colorplots, note however that due to symmetries the actual optima are degenerate.



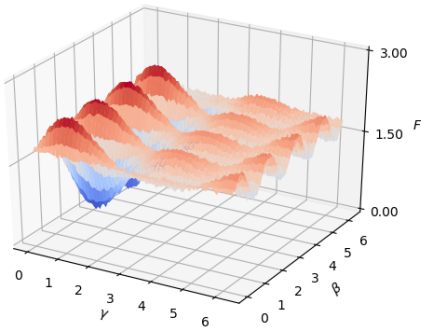
(a) Butterfly graph (unweighted)



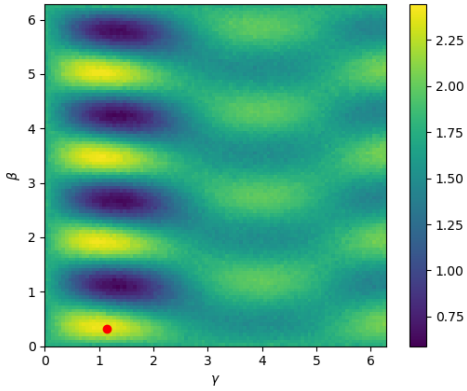
(b) 3D plot of F_1 for the butterfly graph



(c) Colorplot of F_1 for the butterfly graph



(d) 3D plot of F_1 for the weighted butterfly graph



(e) Colorplot of F_1 for the weighted butterfly graph

Figure 3.6: Landscapes of the expectation value F_1 for the unweighted (b), (c) and weighted (d), (e) Butterfly graph using a 3D plot and a colorplot respectively. The graph is visualized in (a). The weights for the weighted graph were drawn from a uniform distribution $w_{ij} \sim U(0,1)$ for the weights of edges $\{i,j\} \in E$. In all plots both γ, β range from 0 to 2π, divided into 100 gridpoints. The expectation value $F_1 = \langle H_C \rangle$ is estimated using 1024 samples from the state $|\gamma, \beta\rangle$ at each gridpoint. The maximum is indicated by a red dot in the colorplots, note however that due to symmetries the actual optima are degenerate.

3.2.4. Adiabatic Theorem and the limit $p \to \infty$

One of the main advantages of QAOA is the fact that its performance monotonically increases with p. This can be shown using the Adiabatic Theorem.

Theorem 3.2.1. *Suppose that the Hamiltonian of a system gradually changes from an initial form H_i to some final form H_f. The Adiabatic Theorem states that if the system was initially in the nth eigenstate of H_i, then at the end of the process, the system will be in the nth eigenstate of H_f [23].*

I would like to emphasize that this change has to be gradual, hence the name adiabatic (very much related to the same term in the context of thermodynamics). Using the Adiabatic Theorem it can be proven that as $p \to \infty$ the QAOA obtains the optimimal solution [20]

$$\lim_{p\to\infty} \max_{\boldsymbol{\gamma},\boldsymbol{\beta}} F_p(\boldsymbol{\gamma},\boldsymbol{\beta}) = \max_{\boldsymbol{x}} C(\boldsymbol{x}) \tag{3.28}$$

This configuration $\boldsymbol{x}$ corresponds to the is the groundstate energy of $-H_C$, where H_C is the cost Hamiltonian defined in (3.2).

3.2.5. Concentration

While it is useful that we are guaranteed to improve the expectation ratio F_p when increasing p, eventually we want to sample from the state $|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle$ to find an approximate solution. For this reason, it is desirable to know if the variance is bounded or not, since we would like to find bit strings with a cutvalue close to F_p or better. This turns out to be the case. In the original paper [20], it was proven that the variance is bounded by the following quantity.

$$\langle\boldsymbol{\gamma},\boldsymbol{\beta}|C^2|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle - \langle\boldsymbol{\gamma},\boldsymbol{\beta}|C|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle^2 \leq 2m\Big[\frac{(v-1)^{2p+2}-1}{(v-1)-1}\Big] \tag{3.29}$$

where v is the maximum degree of the graph and m the number of edges (or clauses). When v and p are taken to be constant, the standard deviation of $C(z)$ is at most of order $\sqrt{m}$. This has the advantageous implication that the sample mean of order m^2 values of $C(z)$ satisfies $|F_p(\boldsymbol{\gamma},\boldsymbol{\beta}) - C(z)| \leq 1$ with probability $1-\frac{1}{m}$, as we can estimate $F_p(\boldsymbol{\gamma},\boldsymbol{\beta})$ with a reasonable amount of samples.

3.3. Finding optimal parameters

Several methods for finding optimal angles have been proposed in [8, 13, 63]. These methods include solving the problem analytically, doing a gridsearch of the parameter space [20], using known optimization methods such as Nelder-Mead [55] or the Broyden–Fletcher–Goldfarb–Shanno algorithm [63], and even machine learning [1, 13].

The first two approaches unfortunately do not scale well with p. Analytically solving for the optimal parameters requires evaluation of subgraphs of depth p. On the other hand, one could use a gridsearch by dividing the parameter space into N points. This approach seems promising for fixed p, as the F-landscape is usually not jagged. To be precise, the partial derivatives of $F_p(\boldsymbol{\gamma},\boldsymbol{\beta})$ were shown to be bounded by $O(m^2+mn)$ [20]. There is one downside to this, to determine $2p$ parameters, we would have to estimate the expectation of the cost function N^{2p} times, as this method is not at all efficient in p. There are also methods that use a variational approach, as will be discussed in the next section.

3.3.1. Variational Quantum Eigensolver

The variational quantum eigensolver (VQE) is a quantum algorithm used to find the groundstate of a given Hamiltonian H [40, 46]. It makes use of the variational principle which states that the expectation value of the Hamiltonian is always greater than or equal to the groundstate energy E_0 [23]

$$\langle H\rangle \geq E_0 \tag{3.30}$$

for any state $|\psi\rangle$. The idea of VQE is to prepare a parameterized state $|\psi(\boldsymbol{\theta})\rangle$ that depends on some parameter set $\boldsymbol{\theta}$. This state is prepared using a unitary operator $U(\boldsymbol{\theta})$.

$$|\psi(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})|0\rangle \tag{3.31}$$

Using the variational principle, the ground state energy can be bound by $\langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle$. The actual goal is to find the optimal set $\boldsymbol{\theta}$ in order to prepare the groundstate, or a state that is close to it

$$\boldsymbol{\theta}_{\text{opt}} = \arg\min_{\boldsymbol{\theta}} \langle\psi(\boldsymbol{\theta})|H|\psi(\boldsymbol{\theta})\rangle \tag{3.32}$$

The optimal parameters can be found with outer-loop classical optimization, see Figure 3.7. Examples of possible methods include Nelder-Mead [42], the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [21] or particle swarm optimization [33, 48].

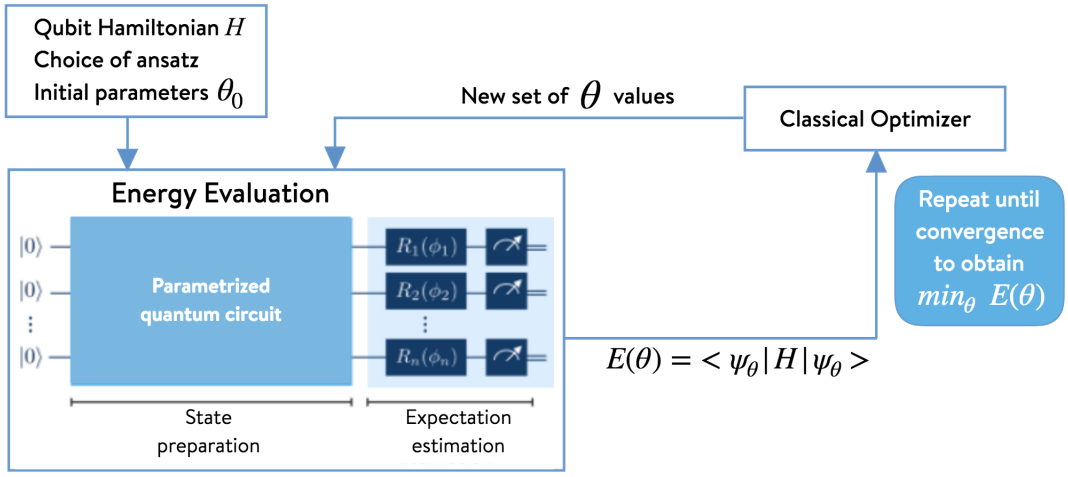


Figure 3.7: Overview of the variational quantum eigensolver. A parametrized state is prepared after which the expectation value of the Hamiltonian is calculated classically. A classical optimization scheme determines a new set of parameters to be tried. This is repeated until convergence, or some other stopping criterium. Figure from [59]

It should be noted that the set of states that are possible to create is restricted by the structure of $U(\boldsymbol{\theta})$. Finding an appropriate ansatz operator $U(\boldsymbol{\theta})$ necessary to find the true groundstate can be difficult without prior knowledge of the problem.

Note that this is very similar to what we have to do in QAOA, namely finding the set of parameters $\boldsymbol{\gamma}, \boldsymbol{\beta}$ to maximize the expectation value of the cost Hamiltonian H_C, which is equivalent to minimizing the expectation value of $-H_C$

$$\boldsymbol{\gamma}_{\text{opt}}, \boldsymbol{\beta}_{\text{opt}} = \arg\min_{\boldsymbol{\gamma},\boldsymbol{\beta}} \langle \boldsymbol{\gamma}, \boldsymbol{\beta}| - H_C |\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle \tag{3.33}$$

The form of the unitary in (3.31) in the case of QAOA is given by

$$U(\boldsymbol{\gamma}, \boldsymbol{\beta}) = U(\beta_p, H_B)U(\gamma_p, H_C)\dots U(\beta_1, H_B)U(\gamma_1, H_C)H^{\otimes n} \tag{3.34}$$

In fact, we can use VQE as a subroutine of QAOA for finding the optimal parameters [60, 63]. An illustration hereof can be found in Figure 3.8. VQE is still an active field of research [4, 12, 58]. It can be applied to QAOA, but also to areas such as chemistry where it is used for finding the groundstate energy of simple molecules [46].

In this thesis I will investigate one of the methods proposed in [63] that makes use of the VQE subroutine. This method exploits certain patterns relating the optimal angles for different layers of the algorithm, reminiscent of the Quantum Adiabatic Algorithm. These patterns can be used to iteratively increase p, and determining an educated guess for the next QAOA layer from which an optimization routine is started for finding the local optimum. A more in depth discussion of these methods can be found in chapter 4.

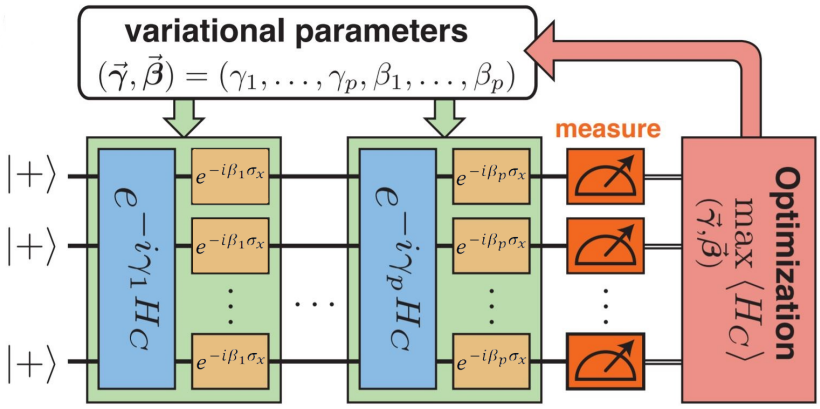


Figure 3.8: Schematic of a p-level Quantum Approximate Optimization Algorithm, making use of the variational approach to find optimal angles. The mixer hamiltonian is split into single qubit gates as explained in Subsection 3.2.1, in theory these can be applied in parallel. Adapted figure from [63].

3.4. Relation to the Quantum Adiabatic Algorithm

The Quantum Adiabatic Algorithm, sometimes referred to as simply Quantum Annealing (QA), is a quantum algorithm for solving the satisfiability problem, proposed in [18] in 2000. The algorithm is based on adiabatic evolution and makes clever use of the adiabatic theorem presented in Section 3.2.4. The evolution of the quantum state is governed by the time-dependent Schrödinger equation

$$i\frac{\partial}{\partial t}|\psi(t)\rangle = H(t)|\psi(t)\rangle \tag{3.35}$$

where I included the factor $\hbar$ in $H(t)$. In QA we use a Hamiltonian $H(t)$ that interpolates between an initial Hamiltonian $H_i = H(0)$, whose ground state is easy to construct, and a final Hamiltonian $H_f = H(T)$, whose ground state encodes the satisfying assignment where T is the total time of the evolution.

$$H_i \xrightarrow{\text{time evolution}} H_f = \sum_{i=1}^{m} h_i \tag{3.36}$$

Oftentimes, one uses an initial Hamiltonian H_i with groundstate $|+\rangle^{\otimes n}$, which is easy to prepare on a quantum computer. A candidate for this initial Hamiltonian is $H_i = -(\sigma_x)^{\otimes n}$, with eigenvalues 1 and -1 corresponding to the eigenstates $|-\rangle^{\otimes n}$ and $|+\rangle^{\otimes n}$, respectively. In the most general case we can represent the evolution using [16, 51]

$$H(t) = f(t)H_i + g(t)H_f \tag{3.37}$$

where $f(t)$ and g(t) are smooth functions of time with boundary conditions $f(0) = g(T) = 1$ and $f(T) = g(0) = 0$. One can simply choose a linear evolution, taking $f(t) = 1 - t/T$ and $g(t) = t/T$, but other options are possible. Now consider the evolution of the state that is subject to this time-dependent Hamiltonian. The solution to (3.35) defines the unitary operator $U(t, t_0)$ [25]

$$|\psi(t)\rangle = U(t, t_0)|\psi(t_0)\rangle \tag{3.38}$$

where t_0 is the initial time. Note that the operator works transitive in the sense that

$$U(t_2, t_0) = U(t_2, t_1)U(t_1, t_0) \tag{3.39}$$

Combining (3.35) and (3.38) yields the following expression

$$i\frac{\partial}{\partial t}U(t, t_0)|\psi(t_0)\rangle = H(t)U(t, t_0)|\psi(t_0)\rangle \tag{3.40}$$

and since the equation must hold for any (properly normalized) $|\psi(t)\rangle$ we find the following partial differential equation

$$i\frac{\partial}{\partial t}U(t, t_0) = H(t)U(t, t_0) \tag{3.41}$$

subject to the initial condition $U(t_0, t_0) = I$ as $|\psi(t)\rangle = |\psi(t_0)\rangle$ if no time has passed. Using a Taylor expansion and substituting the first order temporal derivative with (3.41) we can derive an expression for $U(t + \Delta t, t_0)$ up to second order in Δt

$$U(t + \Delta t, t_0) = U(t, t_0) - iH(t)U(t, t_0)\Delta t + O(\Delta t^2) \tag{3.42}$$

and so by using the initial condition we find

$$U(t + \Delta t, t) = I - iH(t)\Delta t + O(\Delta t^2) = \exp\Big\{-iH(t)\Delta t\Big\} + O(\Delta t^2) \tag{3.43}$$

where the latter equality holds because we are only concerned with terms up to Δt. Using this expression and transitivity, we can derive an expression for $U(t, t_0)$ for arbitrary timesteps $t - t_0$. We do this by dividing the time into very small steps $\epsilon = \frac{t-t_0}{N}$, so equation (3.43) is approximately precise. This yield the following

$$U(t, t_0) = \prod_{k=1}^{N} U(t_0 + k\epsilon, t_0 + (k-1)\epsilon) = \lim_{\epsilon \to 0} \prod_{k=1}^{N} \exp\Big\{-i\epsilon H(t_0 + (k-1)\epsilon)\Big\} \tag{3.44}$$

If we can write this as an exponential of a sum, we might be able to rewrite it as an exponential of an integral when taking the limit $\epsilon \to 0$. However, we have to be wary of non-commutivity. A constant matrix always commutes with itself, but since we are considering a time-dependent matrix it might be the case that $H(t)$ does not commute with itself at different times. If $H(t_i)$ *does* commute with $H(t_j)$ for every pair $t_i, t_j \in [t_0, t]$ we can write $U(t, t_0)$ as

$$U(t, t_0) = \exp\Big\{-i\int_{t_0}^{t} H(t)\,\mathrm{d}t\Big\}, \qquad \text{if } [H(t_i), H(t_j)] = 0 \forall\, t_i, t_j \in [t_0, t] \tag{3.45}$$

Unfortunately, if $H(t_i)$ does *not* commute with $H(t_j)$ for some pair $t_i, t_j \in [t_0, t]$, we need another approach. Instead, we need the **time-ordered exponential** to describe the evolution of the quantum state [25].

$$U(t, t_0) = \mathcal{T} e^{-i\int_{t_0}^{t} \mathrm{d}\tau H(\tau)} \tag{3.46}$$

where

$$\mathcal{T} e^{-i\int_{t_0}^{t} \mathrm{d}\tau H(\tau)} = I + \sum_{k=1}^{\infty} \frac{(-i)^k}{k!} \int_{t_0}^{t} \mathrm{d}t_1 \int_{t_0}^{t_1} \mathrm{d}t_2 \cdots \int_{t_0}^{t_{n-1}} \mathrm{d}t_n H(t_1)H(t_2)\dots H(t_n) \tag{3.47}$$

This integral is hard to evaluate, however we can mitigate this using a the **Suzuki-Trotter** decomposition. In its most basic form we have [27]

$$e^{Ax+Bx} = e^{Ax}e^{Bx} + O(x^2) \tag{3.48}$$

where A and B are arbitrary matrices and x is some parameter. When applying the decomposition, one usually makes use of the relation

$$\Big(e^{\frac{x}{N}A}e^{\frac{x}{N}B}\Big)^N = \underbrace{e^{\frac{x}{N}A}e^{\frac{x}{N}B}\dots e^{\frac{x}{N}A}e^{\frac{x}{N}B}}_{N \text{ times}} = e^{x(A+B)+\frac{1}{2}\frac{x^2}{N}[A,B]+O\left(\frac{x^3}{N^2}\right)} \tag{3.49}$$

Note that we find $e^{(A+B)x} = (e^{\frac{x}{N}A}e^{\frac{x}{N}B})^N$ as N tends to infinity. The Suzuki-Trotter expansion can be used to derive an approximation of the time-ordered exponential by discretizing the elapsed time $t - t_0$ into N small intervals of length Δt [52]

$$U(t, t_0) = \mathcal{T} e^{-i\int_{t_0}^{t} \mathrm{d}\tau H(\tau)} \approx \prod_{k=0}^{N-1} \exp\Big\{-iH(k\Delta t)\Delta t\Big\} \tag{3.50}$$

Taking $H(t)$ from (3.37) we have

$$\prod_{k=0}^{N-1} \exp\Big\{-iH(k\Delta t)\Delta t\Big\} = \prod_{k=0}^{N-1} \exp\Big\{-i\Big(f(k\Delta t)H_i + g(k\Delta t)H_f\Big)\Delta t\Big\} \tag{3.51}$$

and using the approximation (3.48) we find

$$U(t, t_0) \approx \prod_{k=0}^{N-1} \exp\Big\{-i\Delta t f(k\Delta t)H_i\Big\} \exp\Big\{-i\Delta t g(k\Delta t)H_f\Big\} \tag{3.52}$$

The approximation improves as Δt gets smaller, and becomes exact when taking the limit $\Delta t \to 0$. Compare this with QAOA

$$U_{QAOA}(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \prod_{k=1}^{p} \exp\Big\{-i\beta_k H_B\Big\} \exp\Big\{-i\gamma_k H_C\Big\} \tag{3.53}$$

Seeing these parallels, one can consider QAOA as a discrete version of QA. Moreover, in Chapter 4 we will see that for the optimal parameter sets $\boldsymbol{\gamma}, \boldsymbol{\beta}$ we find that γ_i monotonically increases and β_i monotonically decreases, just like linear annealing in QA. These patterns were found both in [63] and [13] independently. A conceptual image of this relation is shown in Figure 3.9

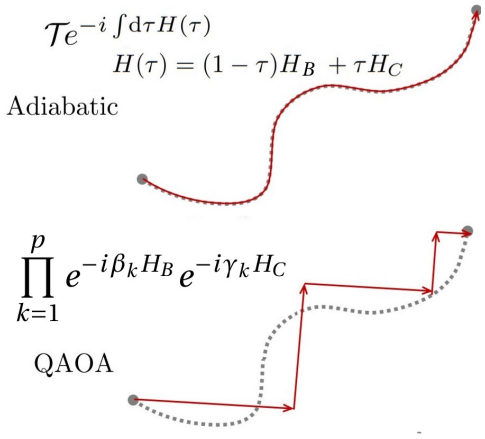


Figure 3.9: Conceptual analogy for comparing (bottom) QAOA to the (top) adiabatic algorithm with linear evolution as a path through state space. Adapted figure from [55]

In order to make sure that the system evolves to the ground state of the final Hamiltonian, the evolution must be gradual, hence the name adiabatic. Therefore, the evolution time T must be long enough. It turns out that the time required for "gradual" evolution depends on the minimum energy gap $\Delta_{\min}$ between the groundstate and the first excited state during the evolution [54].

$$\Delta_{\min} := \inf\Big\{\big|E_1(t) - E_0(t)\big| : t \in [0, T]\Big\} \tag{3.54}$$

here $E_0(t)$ is the groundstate energy of $H(t)$, and similarly $E_1(t)$ is the energy of the first excited state. To guarantee that the system remains in the groundstate, the necessary run time of the algorithm should typically scale as $T = O(1/\Delta^2_{\min})$ [2, 63]. In Figure 3.10 an illustration of the minimum spectral gap is shown.

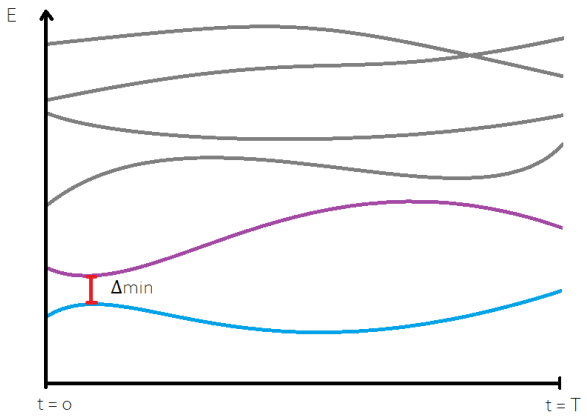


Figure 3.10: Example of an evolution of the eigenenergies of a time-dependent Hamiltonian $H(t)$. The blue line represents the groundstate energy, and the purple line represents the energy of the first excited state. Here $\Delta_{\min}$ denotes the minimum (energy) distance between the groundstate and the first excited state of $H(t)$, as defined in (3.54).

A couple of years after the conception of the algorithm it turned out that for hard instances of 3SAT the gap between the lowest two energies can get exponentially close [19]. More precisely, it was shown that finding the minimum of a classical cost function whose domain is of size N, one needs the runtime to grow with at least the square root of N. In many combinatorial problems the domain size grows exponentially with the problem size n, so the approach is not efficient in n. The best one can hope for is Grover (i.e. quadratic) speed-up, and thus the Adiabatic Algorithm does not disprove $P \neq NP$.

4

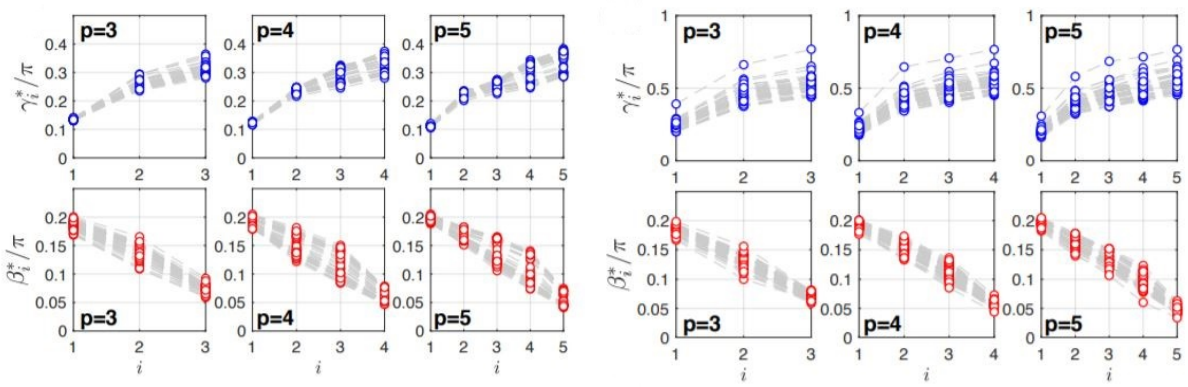
Implementation

The QAOA algorithm is very reliant on finding appropriate angles $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ that favour good partitions and causes bad partitions to destructively interfere. One of the main obstacles to overcome is determining those optimal angles efficiently. In [63] two methods were proposed to tackle this problem, INTERP and FOURIER that run in poly(p) time. In this chapter, I will discuss the INTERP method together with the practical intricacies of implementing QAOA.

All the code developed for this thesis can be found on my GitHub repository: `https://github.com/soosub/bep/` in the folder `Implementation/NewPlan`

4.1. Patterns in the optimal parameters

In order to improve upon current methods, extensive searches for globally optimal angles were conducted in [13, 63] and independently found interesting results. After degeneracies were reduced from the parameter space, there were patterns to be found relating the optimal parameters for a particular graph at level p to the optimal parameters at level $p+1$, as shown in Figure 4.1.



(a) Optimal parameters for unweighted 3-regular graphs (b) Optimal parameters for weighted 3-regular graphs

Figure 4.1: The parameter pattern visualized by plotting the optimal parameters of 40 instances of 16-vertex unweighted 3 regular graphs (a) and 16-vertex weighted 3 regular graphs (b), for $3 \leq p \leq 5$. The weights w_{ij} in the weighted graphs were chosen from a uniform distribution $[0,1]$. Each dashed line connects parameters for one particular graph instance. For each instance and each p, the classical BFGS optimization routine was used from 10^4 random initial points, and the best parameters were kept. Figure and results from [63].

Independently, similar patterns were found on 10 node graphs in [13], see Figure 4.2.

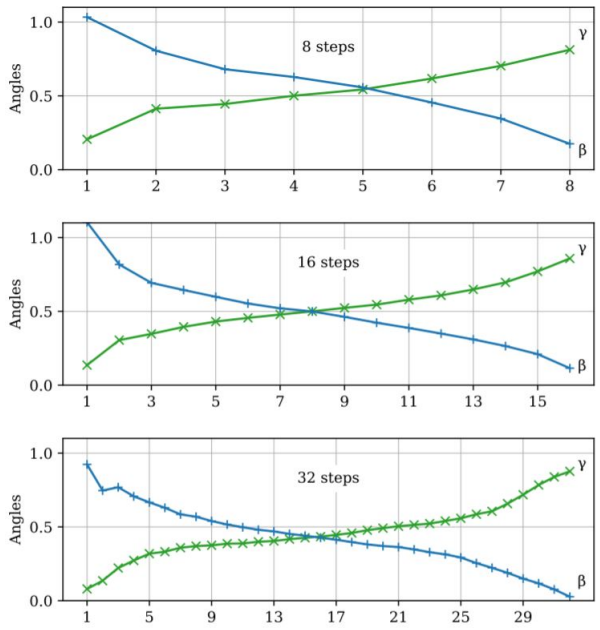


Figure 4.2: Examples of optimized parameters $(\boldsymbol{\beta}_{\text{opt}}, \boldsymbol{\gamma}_{\text{opt}})$ for Max-Cut QAOA on 10 node graphs, with 8, 16, and 32 QAOA steps. The graph was chosen from the Erdős-Rényi ensemble with edge probability 50%. When comparing with Figure 4.1, there is a difference in the scaling of the β_i parameters, this is because [13] used the mixer Hamiltonian $H_B = \frac{1}{2}\sum_j \sigma_x^{(j)}$ whereas [63] used the mixer Hamiltonian $H_B = \sum_j \sigma_x^{(j)}$, resulting in a factor 2 difference. Figure and results from [13].

As one can see from both results, γ_i is a monotonically increasing sequence while β_i is a monotonically decreasing sequence, for $i \in \{1, \dots p\}$. This is reminiscent of the Quantum Adiabatic Algorithm using a linear annealing schedule, where the initial Hamiltonian is turned off gradually while the final, or problem Hamiltonian is turned on [2]. In addition, the optimal parameters found in p-level QAOA are very similar to the optimal parameters in the next level. This opens up the possibility of exploiting this relation in order to speed-up the parameter determination as will be discussed in Section 4.2.

4.2. Proposed method by Zhou et al.

In [63], it was suggested to use the patterns they found in order to enhance the search for optimal parameters. Two methods were proposed to achieve this, namely INTERP and FOURIER. Both methods require poly(p) time, as opposed to, for example, gridsearch. The general idea in both methods is to use the optimal, or quasi-optimal angles at level p, and make an educated guess for the initial point of the optimization routine for the next level $p+1$. In the next section I will focus on the INTERP method.

4.2.1. INTERP

INTERP uses linear interpolation to produce a good initial point $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ for optimising the QAOA parameters as one iteratively increases the level p

$$\left[\boldsymbol{\gamma}^0_{(p+1)}\right]_i = \frac{i-1}{p}\left[\boldsymbol{\gamma}^L_{(p)}\right]_{i-1} + \frac{p-i+1}{p}\left[\boldsymbol{\gamma}^L_{(p)}\right]_i \tag{4.1}$$

$$\left[\boldsymbol{\beta}^0_{(p+1)}\right]_i = \frac{i-1}{p}\left[\boldsymbol{\beta}^L_{(p)}\right]_{i-1} + \frac{p-i+1}{p}\left[\boldsymbol{\beta}^L_{(p)}\right]_i \tag{4.2}$$

for $i = 1, 2, \dots, p+1$. Here, the superscript indicates either that the angle pertains to the local minimum, L, or that it is the initial point for the next optimization round. The subscript inside the brackets indicates the QAOA level, and the subscript outside the brackets indicates the array element. Since the notation might seem a bit daunting, consider the following example. Suppose we have a (local optimal) set $\boldsymbol{\gamma}^L_{(3)} = (\gamma_1, \gamma_2, \gamma_3)$ for $p = 3$ found after some optimization routine. Then, we can find a good initial point $\boldsymbol{\gamma}^0_{(4)} = (\gamma^0_1, \gamma^0_2, \gamma^0_3, \gamma^0_4)$ using the INTERP method for $p = 4$ using these parameters, see Figure 4.3.

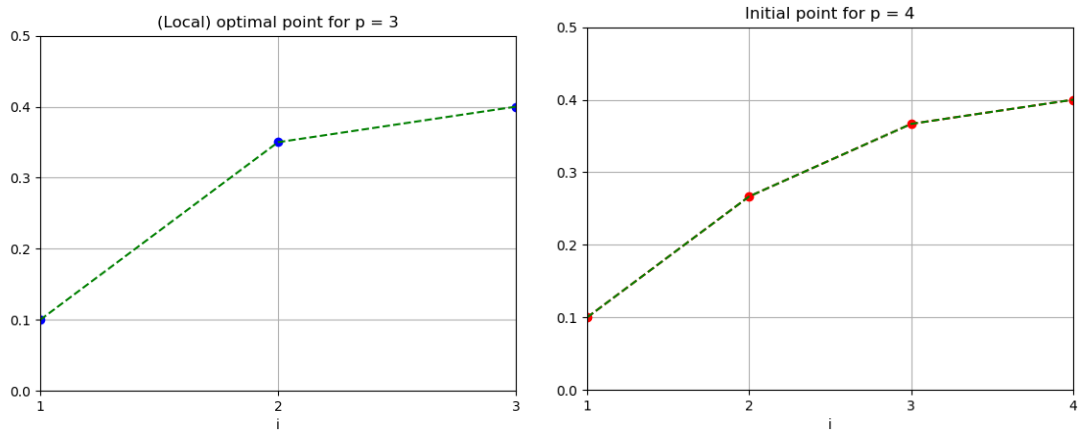


Figure 4.3: Illustration of INTERP method. The method uses the (local) optimal parameters to derive a good initial point for the optimization round of the next layer, using interpolation.

Note that the parameters found using this interpolation are used as *initial point* for the next round of optimization, so these are not directly the parameters used in the circuit. From this initial point we start another round of optimization to find a local optimum, that hopefully is also the global minimum or close to it. This is accomplished by considering the expectation value $\langle H_C \rangle$ as our objective function, which is estimated by sampling multiple times from the state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ and then calculating the corresponding objective value. When the next local minimum is found we use it to generate angles for the next p, and in this fashion we iteratively increase p untill we are satisfied.

In some sense, it is flexible as one is not obliged to start from $p = 1$, but can start at any level p_0 given an initial set of angles $\gamma_1, \beta_1 \ldots \gamma_{p_0}, \beta_{p_0}$. For Max-Cut, it was recommended by Zhou to start from $p_0 = 1$ with $(\gamma, \beta) = (0.8, 0.35)$ as an initial guess and work from there [61, 62]. This is the approach I will use when producing the results in the next chapter. Alternatively, one could choose to do a gridsearch for the first layer requiring $O(1)$ function calls, and then proceed using INTERP.

4.3. Number of samples

One algorithm design choice to consider is the number of samples from $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ one needs when determining the final answer. Taking a constant number of samples would naturally lead to poor performance for large graphs, as the number of possible partitions grows exponentially with n. Another (naieve) option would be to take a certain fraction of the possible number of partitions, but asymtotically this would require $O(2^n)$ function calls. Ideally, we want to bound the number of samples by a polynomial function of n and still get results close to or better than $F_p = \langle H_C \rangle$. It turns out we can. Remember from equation (3.29) that the variance is bounded which implied that for fixed p and v the sample mean of order m^2 values of $C(\mathbf{z})$ will be in $[F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) - 1, F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) + 1]$ with probability $1 - \frac{1}{m}$ [20]. This means for reasonably large graphs ($m \gtrsim 10$), we can simply take m^2 samples and still find our estimate to be between $F_p \pm 1$ with reasonable probability. An illustration of the necessary F_p evaluations as a fraction of the total number of partitions is shown in Figure 4.4.

One thing to notice is that this impairs the applicability of QAOA for small graphs. The algorithm is inherently stochastic and so it requires multiple samples from the state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$. When the number of samples required exceeds the number of possible partitions 2^n this nullifies the benefits of using the algorithm, even if we know good angles beforehand. Note that this fraction is based on *one* evaluation of F_p. If, in addition, we still need to find good or optimal angles using classical optimization in the outer loop, then the number of necessary F_p-evaluations could diminish the usefulness of QAOA. Moreover, if one chooses to use a VQE subroutine for determining optimal angles it is likely that one would need more samples per function call since an unaccurate expectation estimate could restrain the effectiveness of the optimization when looking for (local) minima.

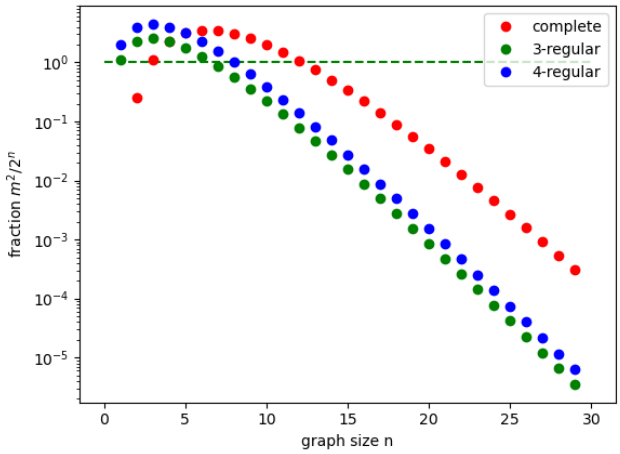


Figure 4.4: The ratio of necessary samples m^2 for finding a reasonalbe estimate for $\langle H_C \rangle$ and the total number of possible partitions 2^n as a function of the graph size n, here m denotes the number of edges in the graph. With a reasonable estimate I mean that we find $F_{p,\text{sampled}}$ within 1 of the actual F_p with a $1-\frac{1}{m}$ probability. The figure shows complete, 3-regular and 4-regular graphs. Note that for small complete graphs brute force approaches are superior to QAOA as one would need more samples then there are possible partitions to accurately estimate F.

4.4. Implementation in Python

4.4.1. QAOA and Goemans-Williamson

In order to examine the INTERP method, I implemented it using Python. For simulating the quantum circuit that is shown in Figure 3.1, I used pyQuil [50]. pyQuil is a quantum instruction language developed at Rigetti Computing. It provides a way to simulate and compile quantum programs. The Quantum Virtual Machine (QVM) was used to run my experiments. Moreover, I made use of Grove, a collection of quantum algorithms built using the Rigetti Forest platform. Grove provides an implementation of QAOA using the VQE subroutine, on top of which I build the INTERP method. This was achieved by running the QAOA circuit for a certain set of initial angles to find a local minimum, which is illustrated in Figure 3.8. This local minimum was then used to calculate a new set of initial angles for the next QAOA layer according to the INTERP scheme. The optimizer used in the VQE subroutine was BFGS [21]. The quantum parts were simulated using the QVM and perfect qubits and full connectivity were assumed. The expectation F_p was calculated during the optimization rounds using a fast method by dotting the wavefunction with the objective operator [50].

To benchmark the INTERP method to other QAOA methods, I also analyzed the performance of QAOA when starting from randomly chosen initial points, again with the Grove's QAOA class. I will abbreviate this method to pyQuil RI (randomly intialised), or simply RI. This means that for a given p, the optimization is simply started from $\boldsymbol{\gamma} = (\gamma_1, ..., \gamma_p)$ and $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)$ with γ_i drawn from a uniform distribution $U[0, 2\pi]$ and β_i from a uniform distribution $U[0, \pi]$. Again I used a BFGS optimizer in the VQE subroutine.

I used the implementation of the Goemans-Williamson algorithm from the package cvxgraphalgs.

4.4.2. Graphs

The graphs that were analyzed were generated using the Python package networkx [26], a Python package for the creation, manipulation, and study of graphs and complex networks. These graphs include cyclic graphs, 3-regular graphs and weighted 3-regular graphs, and graphs from the Erdös-Rényi ensemble with edge probabilities 0.5, and 0.75. To make sure the experiments done are reproducable, I used seeds to keep track of the randomly generated graphs, the 3-regular graphs and the Erdös-Rényi graphs. For the regular graphs, I used a seed ranging from 0 to N, where N is the number of graphs analyzed for that particular class of graphs. The same was done for the Erdös-Rényi (ER) graphs, however for small graphs and low edge probability there is a small probability that there are no edges in the graph. To circumvent this, I simply used the condition that the set of edges had to be non-empty, and if it were empty skip to the next seed.

Python version 3.7.7 was used. Relevant software and Python packages with corresponding versions are included in Appendix D.

5

Results

As mentioned in the previous chapter, for evaluating the performance of the INTERP method I analyzed its performance on several types of graphs, namely cyclic graphs, 3-regular graphs and weighted 3-regular graphs. Additionally, graphs from the Erdös-Rényi ensemble were investigated with edge probabilities 0.5, and 0.75. As a figure of merit of the performance of QAOA approaches, I used the following quantity, as introduced in [63]

$$r = \frac{F_p(\boldsymbol{\gamma}, \boldsymbol{\beta})}{C_{\max}} \tag{5.1}$$

where $C_{\max}$ is the actual optimum. I will refer to the quantity $1 - r$ as the *fractional error*. As the outcome of Goemans-Williamson is inherently stochastic, I introduce the following quantity to compare the Goemans-Williamson on an equal footing as QAOA

$$r_{GW}(k) = \frac{\frac{1}{k}\sum_i^k C_{GW,i}}{C_{\max}} \tag{5.2}$$

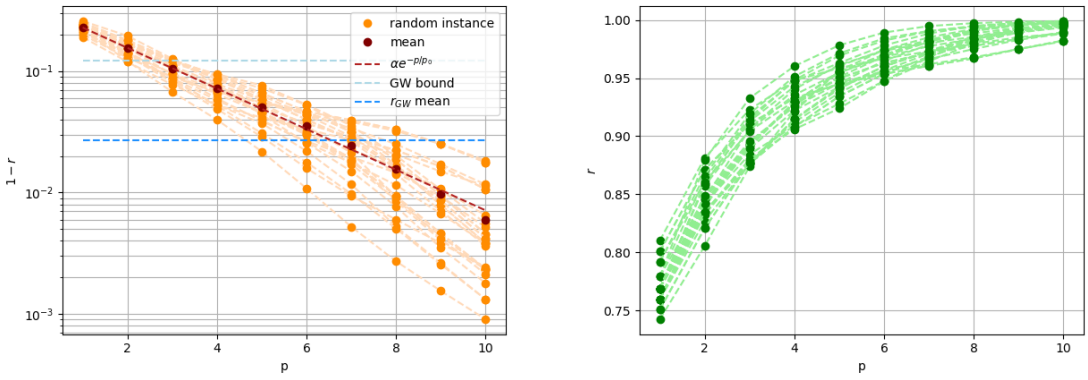
where I take the average outcome of k samples of C_{GW}. In the rest of this chapter I consider $k = 10$.

The results from this chapter were obtained using the pyQuil INTERP method [9], or simply called INTERP, unless stated otherwise. The BFGS optimizer was used for finding the locally optimal parameters $\boldsymbol{\gamma}, \boldsymbol{\beta}$, and by iteratively incrementing p using INTERP starting from $p = 1$ with parameters $(\gamma_0, \beta_0) = (0.8, 0.35)$. Degeneracies from time reversal symmetry $F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) = F_p(-\boldsymbol{\gamma}, -\boldsymbol{\beta})$ were removed by taking $(\boldsymbol{\gamma}, \boldsymbol{\beta})' = (-\boldsymbol{\gamma}, -\boldsymbol{\beta})$ when $\gamma_i, \beta_i < 0$ for all i. Remarkably, for almost all sets of parameters found using INTERP, either all or no angles were found to be negative.

In the appendices the data is presented in more detail. This includes the parameter patterns in Appendix A, the runtime of simulating the algorithm in Appendix B, and the dependence of the fractional error on p in Appendix C.

5.1. 3-regular graphs

The figure of merit r as well as the fractional error, for 25 random instances of unweighted 3-regular graphs with 12 nodes, are plotted in Figure 5.1. Note that the expectation value F_p asymptotically approaches the optimal objective value as we see that r approaches 1 asymptotically. Results for the other graphs are included in Appendix C.



(a) Fractional error $1-r$

(b) Figure of merit r

Figure 5.1: (a) The fractional error $1-r$ for the various randomly generated instances. A fit is included of the form ae^{-p/p_0} where $\alpha = 0.334 \pm 0.003$ and $p_0 = 2.60 \pm 0.03$. (b) r values after optimization for 25 instances of 12 nodal 3-regular graphs at various depths $p = 1,...10$. The angles were obtained by running the Grove QAOA method with the BFGS optimizer, by iteratively incrementing p using INTERP starting from $(\gamma, \beta) = (0.8, 0.35)$.

In Figure 5.2 the patterns of (local) optimal parameters found with the pyQuil INTERP method.

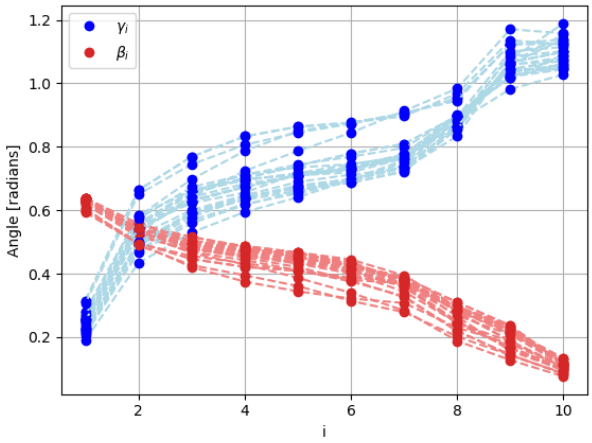
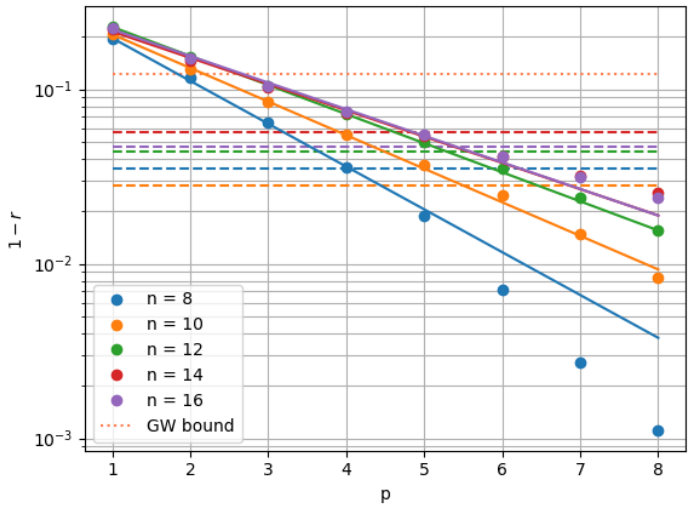


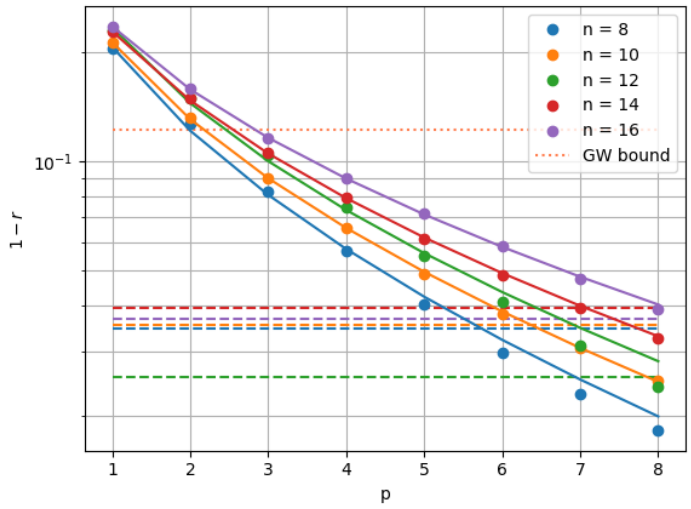
Figure 5.2: Patterns of the angles $\boldsymbol{\gamma}, \boldsymbol{\beta}$ for 25 instances of 12 nodal 3-regular graphs at depth $p = 10$.

These patterns are very reminiscent of the results found in earlier research [13, 63] in Figures 4.2 and 4.1 respectively. Not only does the monotonicity of the patterns coincide, the scaling is also very similar to both, taking the factor 2 in the mixer Hamiltonian into account from [13]. In Appendix A the patterns are shown for different graph sizes n and different classes of graphs. The monotonic patterns seem to be ubiquitous over the different classes of graphs, but there are certainly differences to be seen between the classes. Notable differences include the spread of the patterns for weighted graphs. For unweighted graphs the found parameters for different instances seem to concentrate. Moreover, the range of the patterns differs between the classes. For Erdös-Rényi graphs γ typically varies from $\gamma_1 \approx 0.2$ to $\gamma_p \approx 0.7$ while for 3-regular unweighted graphs we typically find the γ-parameter ranging from $\gamma_1 \approx 0.4$ to $\gamma_p \approx 1$ and for 3-regular weighted graphs $\gamma_1 \approx 0.6$ to $\gamma_p \approx 1.6$.

In Figure 5.3 the relation between p and r for weighted and unweighted 3-regular graphs is shown. As can be seen, for unweighted graphs the fractional error on average decreases exponentially with p. For weighted graphs we see a similar pattern, but for this class the fractional error decreases exponentially with the square root of p. Similar relations were found in [63] for a similar method to INTERP, namely FOURIER.



(a) Unweighted 3-regular graphs

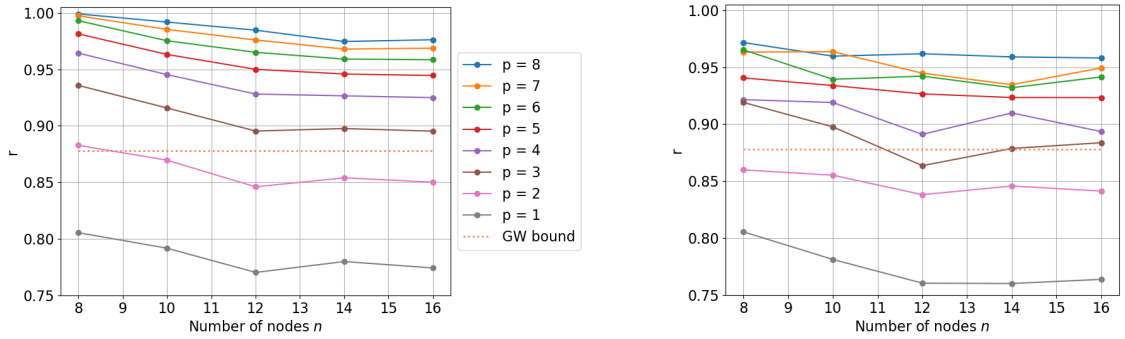


(b) Weighted 3-regular graphs

Figure 5.3: Dependence of r on n on 20 randomly generated unweighted 3-regular (a) and 20 randomly generated weighted 3-regular graphs (b) both using INTERP. For the unweighted graphs the model function ae^{-p/p_0} was used, for the weighted graphs the model function $ae^{-\sqrt{p/p_0}}$ was used, these functions are shown with a solid lines. Moreover, the dashed horizontal lines indicate the average of $1-r_{GW}(10)$ for the 20 instances per nodal number. The coral coloured dotted line indicates $1-\rho$, where $\rho \approx 0.878$ is the approximation ratio of Goemans-Williamson.

It is worth noting that from $p = 6$ onwards QAOA outperforms Goemans-Williamson on all unweighted graphs when comparing the figures of merit r and r_{GW}. For weighted 3-regular graphs we see that $p = 8$ is not sufficient to outperform Goemans-Williamson on graphs of size 12 and 16. For this class of graph Goemans-Williamson often significantly performs better than the lower bound of 0.878 in terms of the figure of merit r_{GW}.

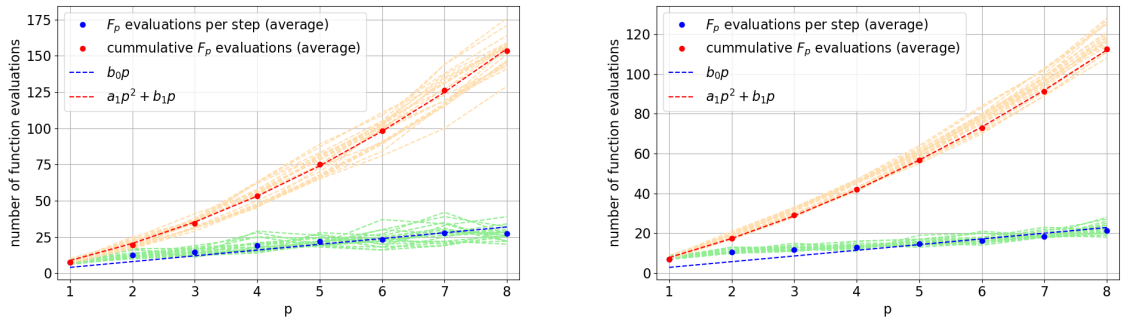
A graph of the dependence of r on the number of nodes n for unweighted 3-regular graphs is shown in Figure 5.4a. As we can see, generically r decreases with n for $p \geq 4$ for the INTERP method. For lower p there seems to be some anomaly for 12 nodal graphs. In addition, we see that INTERP outperforms RI significantly for $p \geq 3$, especially on graphs with more nodes.



(a) INTERP (b) RI

Figure 5.4: Dependence of r on n on 20 randomly generated unweighted 3-regular graphs using both pyQuil INTERP (a) as well as pyQuil RI (b). In both figures the lower bound of 0.878 for the GW-algorithm is included.

The number of function evaluations necessary for parameter determination was also analyzed, see Figure 5.5. The numerical results suggest that indeed the method is polynomial in p. For 3-regular graphs it was found that the necessary number of function evaluations was less than the number of function evaluations required for finding parameters for unweighted graphs.

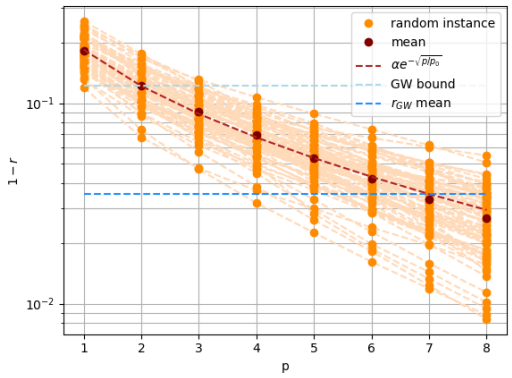


(a) Unweighted (b) Weighted

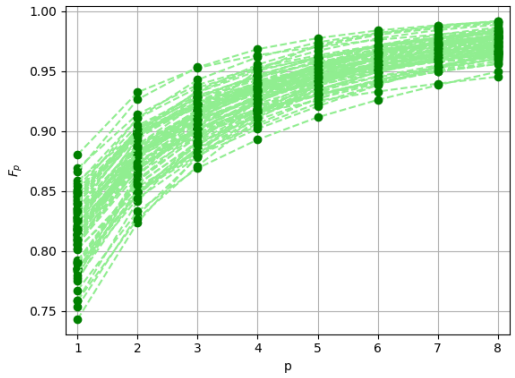
Figure 5.5: Number of function evaluations necessary for parameter optimization in INTERP for 20 instances of 16-nodal 3-regular graphs. Both unweighted (a) and weighted (b). The fit parameters for the mean of the unweighted graphs are $b_0 = 4.0 \pm 0.2, a_1 = 1.52 \pm 0.05, b_1 = 7.2 \pm 0.3$. For the mean of the weighted graph the fitparameters found were $b_0 = 2.8 \pm 0.2, a_1 = 0.89 \pm 0.02, b_1 = 6.9 \pm 0.1$.

5.2. Erdös-Rényi graphs

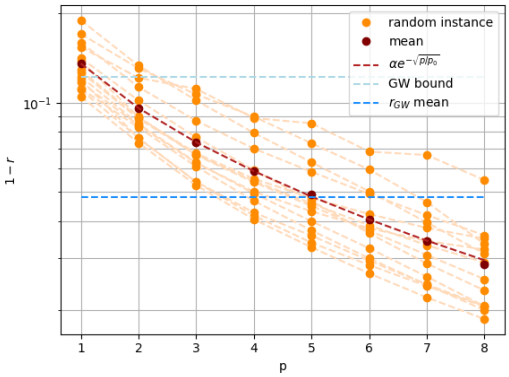
I also examined the performance of QAOA on Erdös-Rényi (ER) graphs with edge probability 0.50 and 0.75, abbreviated as ER-0.50 and ER-0.75 respectively. For graphs of this ensemble we find that the dependence of the fractional error acts similar to that of weighted 3-regular graphs as it also decreases exponentially with the square root of p, see Figure 5.6.



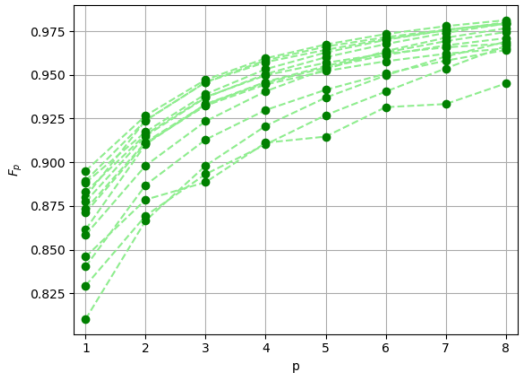
(a) Fractional error $1-r$ for ER-0.50 graphs (20 instances). The fit parameters are $\alpha = 0.51 \pm 0.01$ and $p_0 = 0.99 \pm 0.03$.



(b) Figure of merit r for ER-0.50 graphs (20 instances).



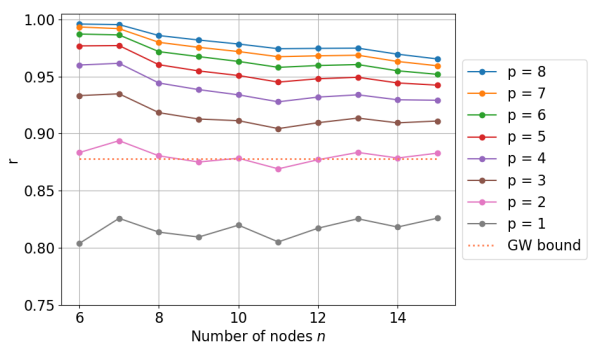
(c) Fractional error $1-r$ for ER-0.75 graphs (10 instances). The fit parameters are $\alpha = 0.164 \pm 0.007$ and $p_0 = 4.1 \pm 0.3$



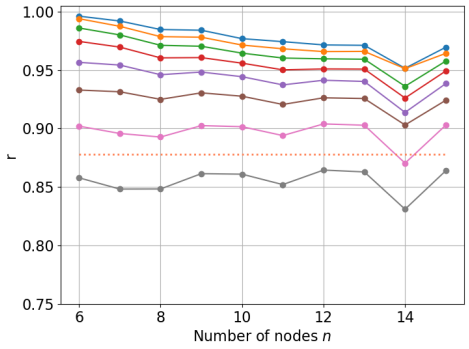
(d) Figure of merit r for ER-0.75 graphs (10 instances)

Figure 5.6: The fractional error and figure of merit for 12 nodal graphs of the Erdös-Rényi ensemble with edge probabilities 0.50 (a), (b) and 0.75 (c), (d) using the INTERP method for finding the parameters.

The dependence of r on the number of nodes n for the Erdös-Rényi graphs is shown in Figure 5.7. Again, we observe a general decreasing pattern in r with n from $p = 4$ onwards.



(a) ER-0.50

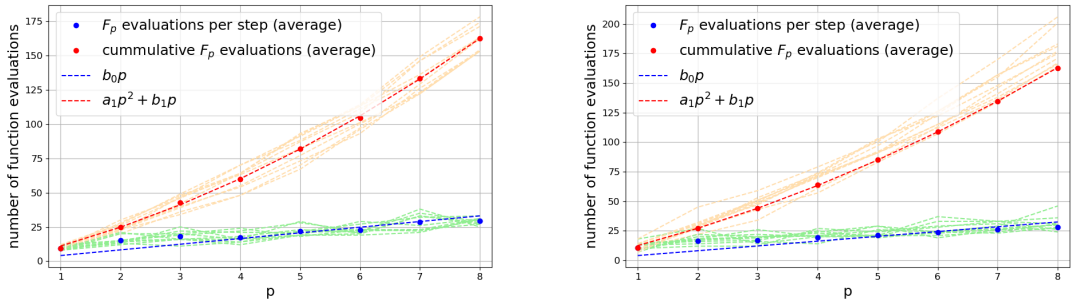


(b) ER-0.75

Figure 5.7: Dependence of r on n on randomly generated unweighted ER graphs with edge probability 0.50 (a) as well 0.75 (b). In both figures the lower bound of 0.878 for the GW-algorithm is included.

The number of function evaluations for parameter determination is shown in figure 5.8. We again observe

that the method is polynomial in p. There seems to be no significant difference in the number of required function evaluations when comparing ER-0.50 and ER-0.75 graphs.



(a) 15 nodal ER-0.50 graphs (20 instances)

(b) 15 nodal ER-0.75 graphs (10 instances)

Figure 5.8: Number of F_p function evaluations necessary for the INTERP method to converge on ER graphs with 15 nodes. (a) 20 instances with edge probability 0.5 and fit parameters $b_0 = 4.1 \pm 0.3$ and $a_1 = 1.32 \pm 0.04, b_1 = 9.7 \pm 0.3$ (b) 10 instances with edge probability 0.75 and fit parameters $b_0 = 4.5 \pm 0.4$ and $a_1 = 1.25 \pm 0.03, b_1 = 12.4 \pm 0.2$.

5.3. Cyclic graphs

In the original QAOA paper [20], an analytical expression for F_p was derived by analyzing the contribution of subgraphs of size p to the expectation value. As the graphs are cyclic, for $p < n/2$ there is only one type of subgraph, namely, a linear array of nodes. After maximizing the function numerically for $p = 1, 2, 3, 4, 5$ and 6 it was conjectured that for cyclic graphs one finds

$$\max_{\boldsymbol{\gamma},\boldsymbol{\beta}} F_p = M_p = n\frac{2p+1}{2p+2} \tag{5.3}$$

for all $p < n/2$. To test this conjecture, I applied the pyQuil INTERP method to cyclic graphs. The F_p values found by this method is shown in Figure 5.9.

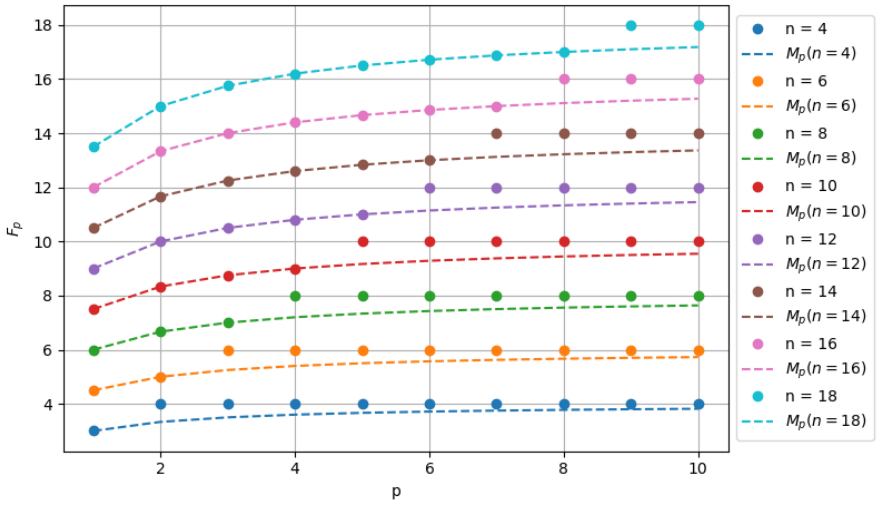
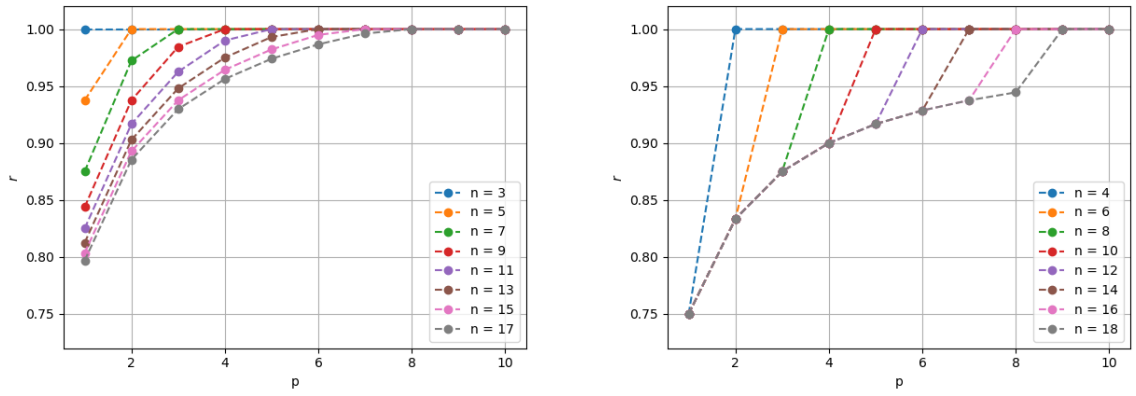


Figure 5.9: The expectation value F_p using the pyQuil INTERP method, for cyclic graphs with an even number of nodes n. The dots indicate the actual F_p values from 1024 samples of the state. The dashed lines indicates the $M_p(n)$ values from Equation (5.3) for various n as a function of p.

It is noteworthy that for $p < n/2$ we indeed find that M_p bounds F_p, and using the pyQuil INTERP method we attain that maximum M_p with high precision. Beyond that, for $p \geq n/2$, the expectation value F_p jumps to the facinity of the optimum value, namely $C_{\max} = n$, meaning $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ is a groundstate of $-H_C$.

In Figure 5.10 the figure of merit r is shown for cyclic graphs. As can be seen, there is a stark contrast between cyclic graphs with an even number of nodes, and an odd number of nodes. This is because the latter group has more partitions that are optimal as there has to be one neighbouring pair that are on the same side of the partition. Cyclic groups with an even number of nodes simply have one groundstate partitions, up to $\mathbb{Z}_2$ symmetry, namely an alternating bit string. This means that the groundstate of $-H_C$ has a higher degeneracy and thus the parametrized state $|\boldsymbol{\gamma}, \boldsymbol{\beta}\rangle$ naturally overlaps more with the groundstate.



(a) Odd n where $C_{\max} = n - 1$

(b) Even n where $C_{\max} = n$

Figure 5.10: Figure of merit r for cyclic graphs of sizes $n = 2, \ldots, 18$. The values for F_p are acquired using the angles obtained using the pyQuil INTERP method.

6

Conclusions and Future Work

The Quantum Approximate Optimization Algorithm (QAOA) is a variational quantum algorithm designed to solve combinatorial optimization problems that depends on $2p$ parameters. Due to its shallow depth it is relatively robust against errors and thus surpasses the need for error correction, making it an interesting algorithm to test on NISQ devices. One of the challenges for feasible use of the algorithm is the determination of suitable parameters. In this thesis I implemented and evaluated the INTERP method proposed by Zhou et al. [63] that exploits similarities with the Quantum Adiabatic Algorithm in order to find quasi-optimal parameters in poly(p) time. This was realized by benchmarking it against the classical Goemans-Williamson algorithm on a variety of graph classes, namely cyclic, 3-regular weighted and unweighted, and Erdös-Rényi graphs with edge probability 0.5 and 0.75.

It was found that the depth p required for surpassing the performance of Goemans-Williamson was dependent on the graph class. While only $p = 2$ and in some cases $p = 3$ was required for QAOA to surpass the approximation ratio of 0.878, the Goemans-Williamson algorithm is often able to do better. Accordingly, a different metric was chosen, namely r_{GW} to compare Goemans-Williamson on equal footing with QAOA. For $p \gtrsim 6$ for unweighted 3-regular graphs, and $p \gtrsim 7$ for weighted 3-regular graphs, the QAOA algorithm using the INTERP method consistently outperforms the classical Goemans-Williamson algorithm in terms of this measure for small graphs with $n \lesssim 16$. Similarly, for Erdös-Rényi graphs $p \gtrsim 7$ was required for graphs with edge probability 0.5 and edge probability 0.75. However, this figure is also dependent on the graph size.

One important aspect of the algorithm is the amount of objective function calls required to find the parameters. It was found that this number is significant to the number of partitions for the small graphs analyzed in this thesis. This leads me to conclude that in practice one should prefer Goemans-Williamson over QAOA for small graphs when approximate solutions suffice.

The circuit implementing the algorithm applied to the Max-Cut problem requires only $O(n^2)$ gates by design, and the numeric results from this thesis suggested that the amount of function calls required for the outer loop classical optimization only grows polynomially in p, as was claimed in [63]. Therefore, the computational cost of finding quasi-optimal parameters using INTERP is expected to only grow polynomially in n and p. For large graph sizes, the number of function calls will no longer be significant to the number of possible partitions as the latter grows exponentially with the problem size.

Moreover, it was found that monotonic patterns in the parameters were consistenly observed in all the classes of graphs that were investigated. One might wonder if this was enforced by the INTERP method itself. While it is possible that the found parameters are local optima, the results also show that the expectation value F_p monotonically increases with p. This fact illustrates that the heuristic works well and offers a good alternative to gridsearches or randomly initialised optimization routines given the costly probing of the parameter space.

The work conducted in this thesis has also led to new questions. One of the main topics that requires further examination is whether or not it is possible to speed up the parameter determination. One idea that comes to mind is to use optimal parameters for one instance of a class of graphs, and use this to find good partitions. Closely related to this idea is to construct some function that closely resemebles the optimum pattern of a given graph class, and use this function to determine parameters, or start optimizing from this point to find local optimal parameters. Possibly, this could offer an advantage over an iterative method like INTERP

that starts from a low p_0 and whose performance is greatly influenced by its initial point $(\boldsymbol{\gamma}_{(p_0)}, \boldsymbol{\beta}_{(p_0)})$. This approach could be aided with Machine Learning techniques to recognize relevant features of a given graph and its class.

Another topic of interest for future research is the implementation on actual quantum devices. As simulating the QAOA circuit on a classical device is not time-efficient, it is hard to get intuition for the time it takes on real quantum hardware combined with a classical computer. Because we are now approaching quantum devices with around 100 bits, it would be interesting to see if QAOA offers an advantage over classical algorithms for large problem sizes, as the exponential nature of many combinatorial optimization problems really becomes relevant for high n. A possible choke point for the speed of the algorithm would be the communication between the quantum computer and the classical computer. It would be interesting to see if hardware especially dedicated to these hybrid algorithms has advantageous beyond classical capability.

By extension, the problems that are forthcoming from real devices should also be investigated, such as the algorithm's resistance to error. Moreover, the mapping of the circuit onto the processor is subject to the processor constraints which might cause a loss in performance as real processors have limitted qubit connectivity. Furthermore, it should be investigated how the sampled average affects the speed of the classical optimization routine as it might cause problems due to the stochastic nature of the measurement outcomes.

Bibliography

[1] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh. Accelerating Quantum Approximate Optimization Algorithm using Machine Learning. *arXiv e-prints*, art. arXiv:2002.01089, February 2020.

[2] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Rev. Mod. Phys.*, 90:015002, January 2018. doi:10.1103/RevModPhys.90.015002.

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[4] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, et al. Quantum Approximate Optimization of Non-Planar Graph Problems on a Planar Superconducting Processor. *arXiv e-prints*, art. arXiv:2004.04197, April 2020.

[5] Abraham Asfaw, Stephen Wood, James Wootton, et al. Learn Quantum Computation Using Qiskit, 2020. URL `http://community.qiskit.org/textbook`.

[6] Boaz Barak and David Steurer. Proofs, beliefs, and algorithms through the lens of sum-of-squares, 2016. URL `https://www.sumofsquares.org/public/index.html`.

[7] Piotr Berman and Marek Karpinski. On some tighter inapproximability results. In Jiří Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming*, pages 200–209, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48523-0.

[8] Fernando G. S. L. Brandao, Michael Broughton, Edward Farhi, Sam Gutmann, and Hartmut Neven. For Fixed Control Parameters the Quantum Approximate Optimization Algorithm's Objective Function Value Concentrates for Typical Instances. *arXiv e-prints*, art. arXiv:1812.04170, December 2018.

[9] Joost Bus. Source code of the QAOA pyQuil INTERP method, 2020. URL `https://github.com/soosub/bep/tree/master/Implementation`.

[10] Jin-Yi Cai. CS 810: Introduction to complexity theory - lecture 20: Goemans-Williamson MAXCUT approximation algorithm, March 2003. URL `http://pages.cs.wisc.edu/~jyc/02-810notes/lecture20.pdf`.

[11] Jin-Yi Cai. CS 810: Introduction to complexity theory - lecture 19: Randomized algorithms: MAXCUT, March 2003. URL `http://pages.cs.wisc.edu/~jyc/02-810notes/lecture19.pdf`.

[12] Marco Cerezo, Kunal Sharma, Andrew Arrasmith, and Patrick J. Coles. Variational Quantum State Eigensolver. *arXiv e-prints*, art. arXiv:2004.01372, April 2020.

[13] Gavin E. Crooks. Performance of the Quantum Approximate Optimization Algorithm on the Maximum Cut Problem. *arXiv e-prints*, art. arXiv:1811.08419, November 2018.

[14] George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[15] Tobias Dantzig. *Numbers: The Language of Science*. Macmillan, 1930.

[16] Saurya Das, Randy Kobes, and Gabor Kunstatter. Energy and efficiency of adiabatic quantum search algorithms. *Journal of Physics A: Mathematical and General*, 36(11):2839, 2003.

[17] Edward Farhi and Aram W. Harrow. Quantum Supremacy through the Quantum Approximate Optimization Algorithm. *arXiv e-prints*, art. arXiv:1602.07674, February 2016.

[18] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum Computation by Adiabatic Evolution. *arXiv e-prints*, art. quant-ph/0001106, January 2000.

[19] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Daniel Nagaj. How to Make the Quantum Adiabatic Algorithm Fail. *arXiv e-prints*, art. quant-ph/0512159, December 2005.

[20] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A Quantum Approximate Optimization Algorithm. *arXiv e-prints*, art. arXiv:1411.4028, November 2014.

[21] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[22] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42(6):1115–1145, November 1995.

[23] David J. Griffiths. *Introduction to Quantum Mechanics*. Pearson, second edition, 2015. ISBN 978-93-325-4289-1.

[24] Lov K. Grover. A fast quantum mechanical algorithm for database search. *arXiv e-prints*, art. quant-ph/9605043, May 1996.

[25] Howard Haber. Physics 215: Time Ordered Exponential, dec 2018. URL `http://scipp.ucsc.edu/~haber/ph215/TimeOrderedExp.pdf`.

[26] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. pages 11–15, August 2008.

[27] Naomichi Hatano and Masuo Suzuki. *Finding Exponential Product Formulas of Higher Orders*, volume 679, page 37. 2005. doi:10.1007/11526216_2.

[28] Jack D. Hidary. *Quantum Computing: An Applied Approach*. Springer, 2019.

[29] Kevin Hollbach and Lilly Li. Lecture - Goemans and Williamson Algorithm for MAXCUT, February 2018.

[30] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, July 2001. ISSN 0004-5411. doi:10.1145/502090.502098.

[31] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[32] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972. ISBN 978-1-4684-2001-2. doi:10.1007/978-1-4684-2001-2_9.

[33] James Kennedy and Russell C. Eberhart. Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, IV:1942–1948, 1995. doi:10.1109/ICNN.1995.488968.

[34] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 767–775, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581134959. doi:10.1145/509907.510017.

[35] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM Journal on Computing*, 37(1):319–357, 2007. doi:10.1137/S0097539705447372.

[36] Erica Klarreich. Approximately hard: The unique games conjecture, October 2011. URL `https://www.simonsfoundation.org/2011/10/06/approximately-hard-the-unique-games-conjecture/`. Retrieved June 17, 2020.

[37] Loszlo Lovasz, Martin Grotschel, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*. Springer, 1988.

[38] George B. Mathews. On the partition of numbers. *Proceedings of the London Mathematical Society*, 28: 486–490, 1897.

[39] Andy Matuschak and Michael A. Nielsen. Quantum computing for the very curious, 2019. URL https://quantum.country/qcvc.

[40] Jarrod McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, February 2016. doi:10.1088/1367-2630/18/2/023023.

[41] Miym. Max-cut, 2009. URL https://commons.wikimedia.org/wiki/File:Max-cut.svg. Retrieved May 18, 2020.

[42] John A. Nelder and Roger Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, January 1965. ISSN 0010-4620. doi:10.1093/comjnl/7.4.308.

[43] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[44] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[45] Gábor Pataki. Cone-LP's and semidefinite programs: Geometry and a simplex-type method. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 162–174. Springer, 1996.

[46] 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*, 5:4213, July 2014. doi:10.1038/ncomms5213.

[47] John Preskill. Quantum Computing in the NISQ era and beyond. *arXiv e-prints*, art. arXiv:1801.00862, January 2018.

[48] Yuhui Shi and Russell C. Eberhart. A modified particle swarm optimizer. *Proceedings of IEEE International Conference on Evolutionary Computation*, page 69–73, 1998. doi:10.1109/ICEC.1998.699146.

[49] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[50] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture. *arXiv e-prints*, art. arXiv:1608.03355, August 2016.

[51] Jie Sun, Song-Feng Lu, and L. Braunstein Samuel. On Models of Nonlinear Evolution Paths in Adiabatic Quantum Algorithms. *Communications in Theoretical Physics*, 59(1):22–26, January 2013. doi:10.1088/0253-6102/59/1/05.

[52] Yin Sun, Jun-Yi Zhang, Mark S. Byrd, and Lian-Ao Wu. Adiabatic Quantum Simulation Using Trotterization. *arXiv e-prints*, art. arXiv:1805.11568, May 2018.

[53] Pravin M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual Symposium on Foundations of Computer Science*, pages 338–343, 1989. doi:10.1109/SFCS.1989.63500.

[54] Wim van Dam, Michele Mosca, and Umesh Vazirani. How Powerful is Adiabatic Quantum Computation? *arXiv e-prints*, art. quant-ph/0206003, May 2002.

[55] Guillaume Verdon, Michael Broughton, and Jacob Biamonte. A quantum algorithm to train neural networks using low-depth circuits. *arXiv e-prints*, art. arXiv:1712.05304, December 2017.

[56] Sascha S. Wald. *Thermalisation and Relaxation of Quantum Systems*. PhD thesis, Institute Jean Lamour, 2017.

[57] Rui-Sheng Wang and Li-Min Wang. Maximum cut in fuzzy nature: Models and algorithms. *Journal of computational and applied mathematics*, 234(1):240–252, 2010.

[58] Andreas Woitzik, Panagiotis Barkoutsos, Filip Wudarski, Andreas Buchleitner, and Ivano Tavernelli. Entanglement production and convergence properties of the variational quantum eigensolver. *arXiv: Quantum Physics*, 2020.

[59] Takeshi Yamazaki, Shunji Matsuura, Ali Narimani, Anushervon Saidmuradov, and Arman Zaribafiyan. The Variational Quantum Eigensolver (VQE), 2019. URL http://openqemist.1qbit.com/docs/vqe_microsoft_qsharp.html. Retrieved May 18, 2020.

[60] Jiahao Yao, Marin Bukov, and Lin Lin. Policy gradient based quantum approximate optimization algorithm. *arXiv e-prints*, abs/2002.01068, 2020.

[61] Leo Zhou. genQAOA, 2019. URL https://github.com/leologist/GenQAOA.

[62] Leo Zhou. Private communication, May 2020.

[63] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D. Lukin. Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices. *arXiv e-prints*, art. arXiv:1812.01041, December 2018.

A

Parameter patterns

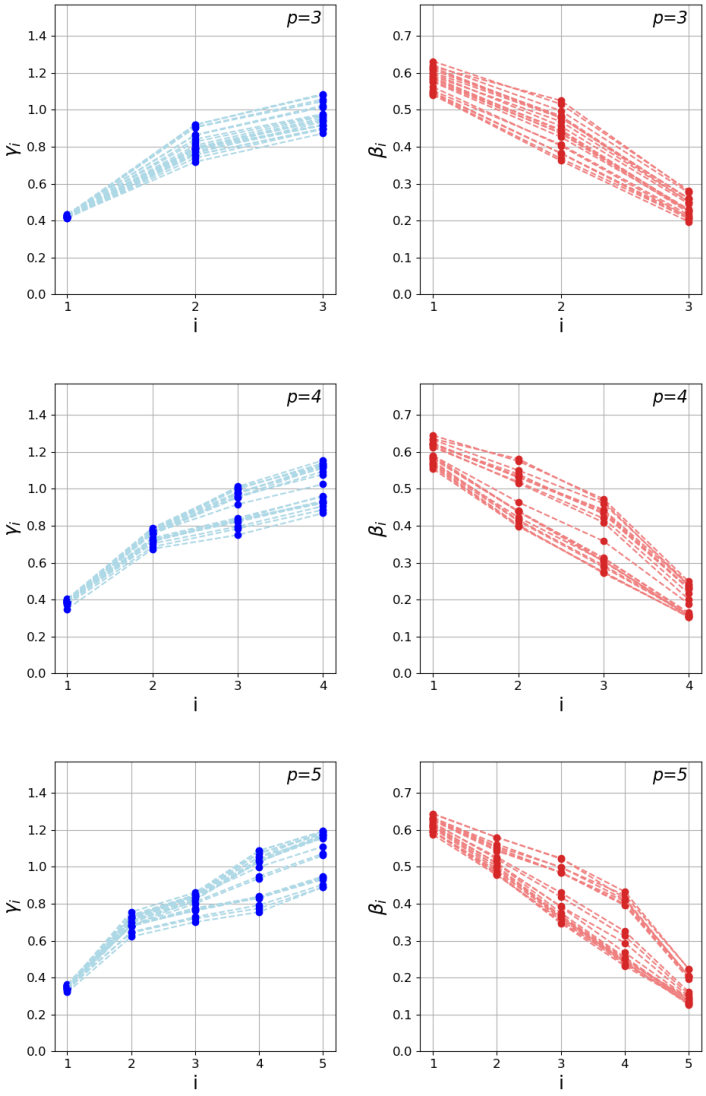


Figure A.1: Quasi-optimal parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ found using the pyQuil INTERP method for 20 instances of 3-regular unweighted graphs with 16 nodes for $p = 3, 4, 5$.

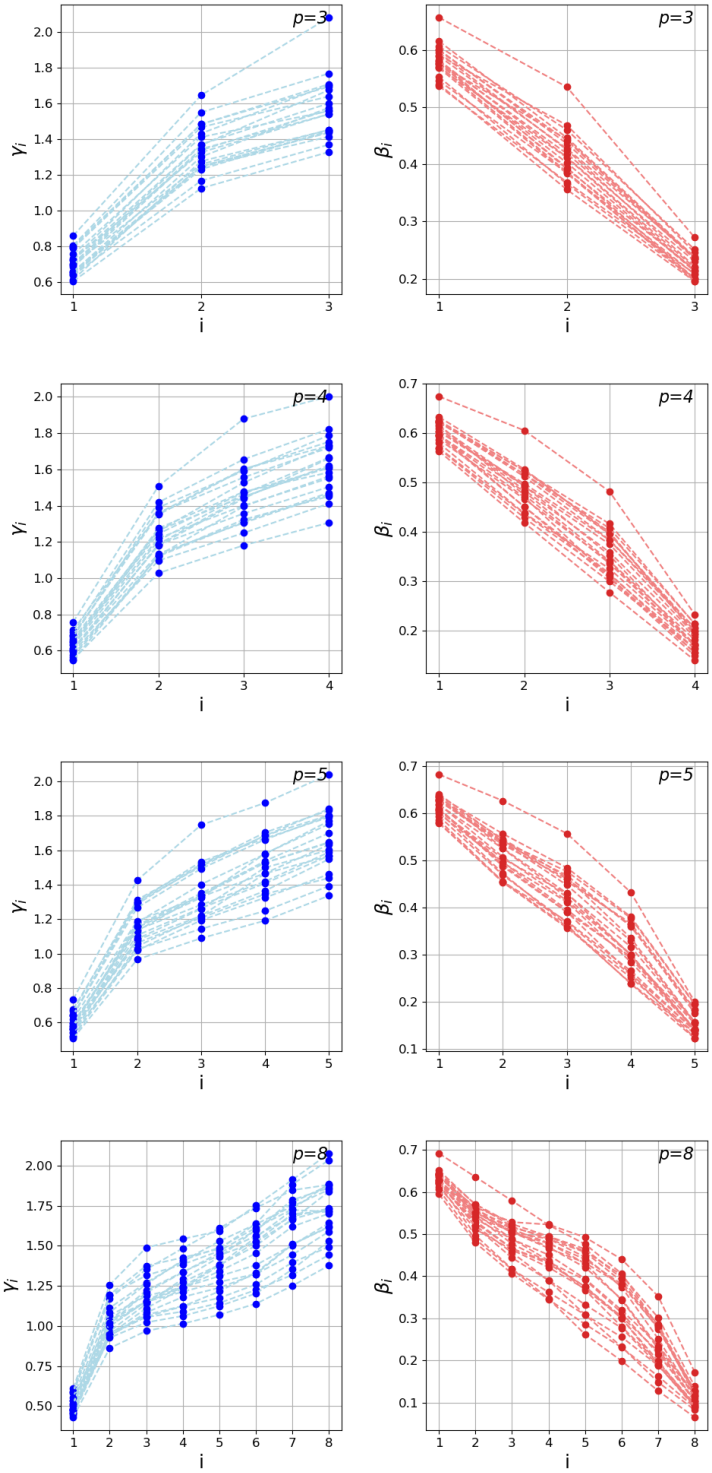


Figure A.2: Quasi-optimal parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ found using the pyQuil INTERP method for 20 instances of 3-regular weighted graphs with 16 nodes for $p = 3, 4, 5$ and 8

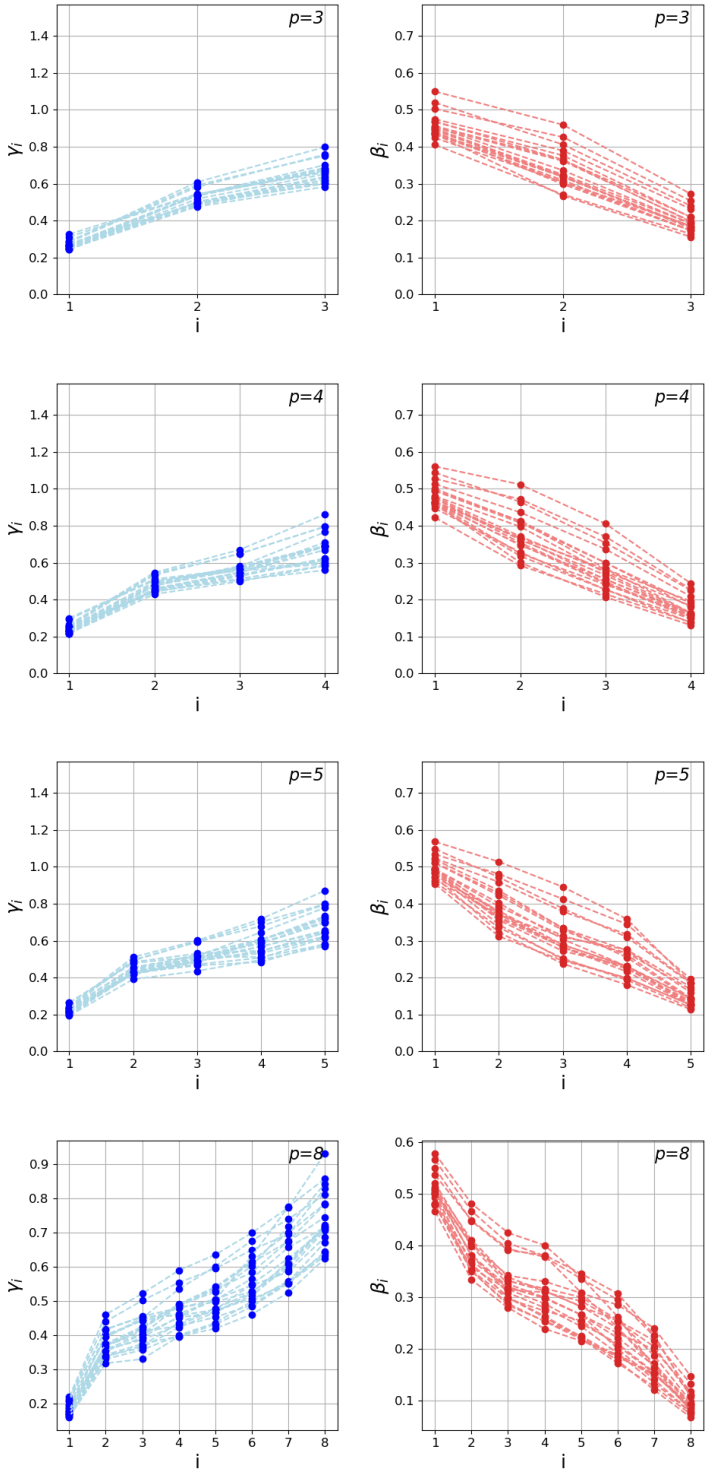


Figure A.3: Quasi-optimal parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ found using the pyQuil INTERP method for 20 instances of 12-nodal unweighted graphs drawn from the Erdős-Rényi ensemble with edge probability 0.5 for $p = 3, 4, 5$ and 8.

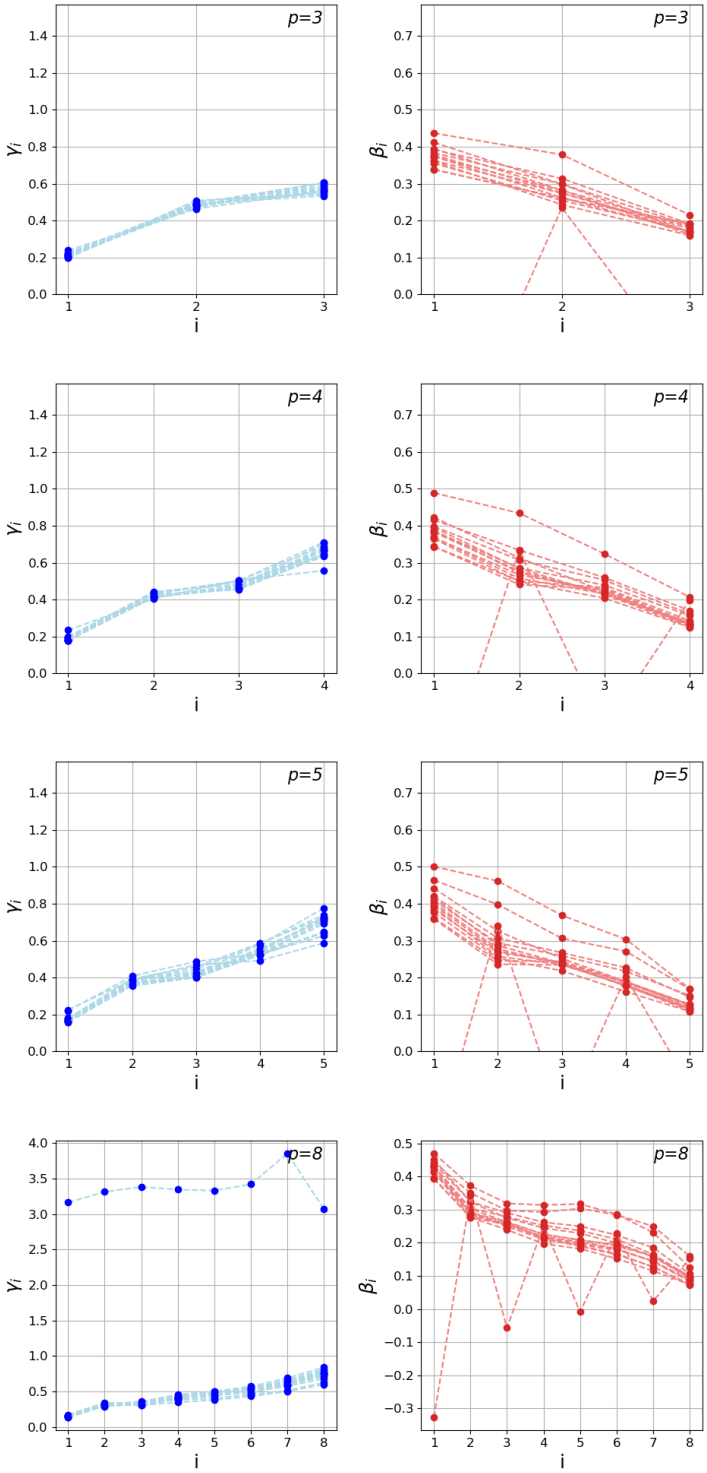


Figure A.4: Quasi-optimal parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ found using the pyQuil INTERP method for 10 instances of 12-nodal unweighted graphs drawn from the Erdős-Rényi ensemble with edge probability 0.75 for $p = 3, 4, 5$ and 8. There seems to be one graph with anomalous results for which the β_i parameter oscillates around 0 and the γ_i parameters lie around π. This is best seen on the graph for $p = 8$, where I did not restict the limits of the vertical axis. Possibly this anomaly has to do with the degeneracy of the optima, as I only removed the degeneracies due to time-reversal symmetry.

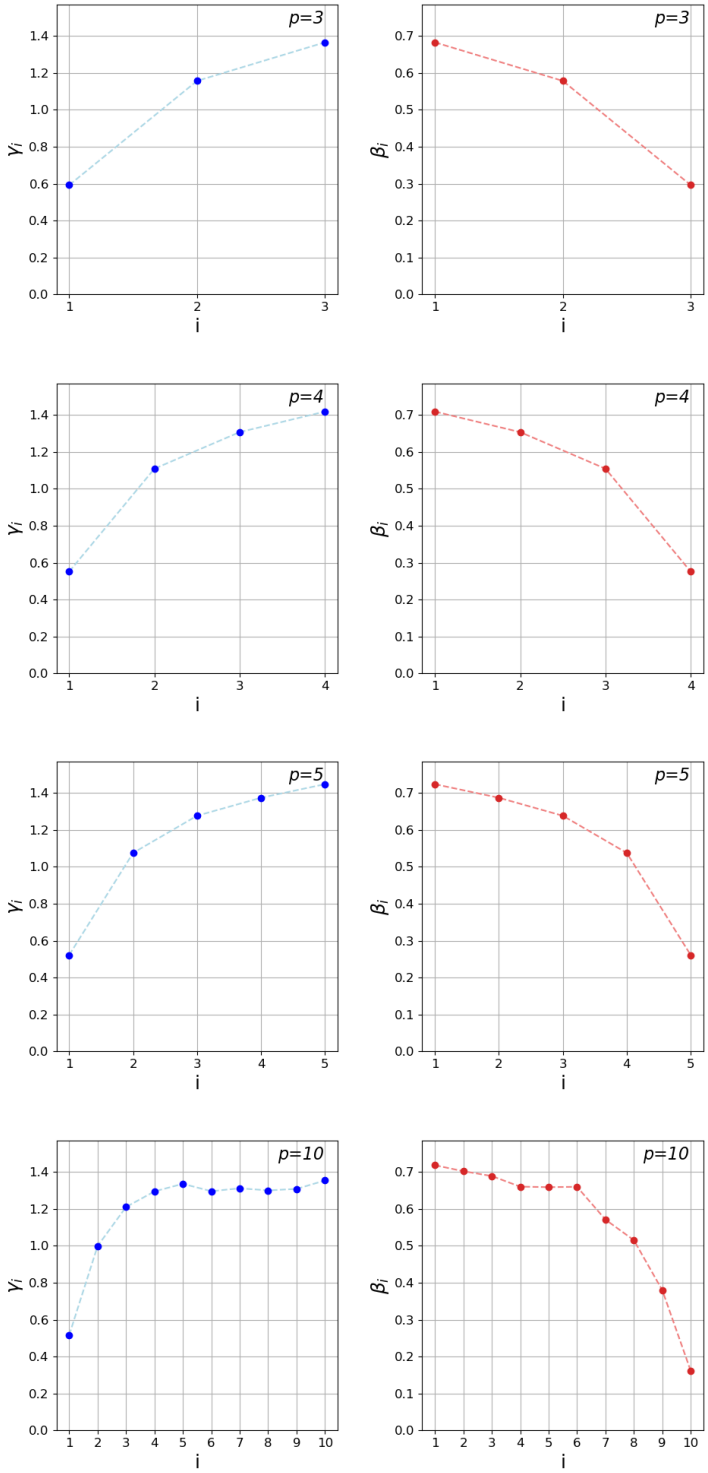


Figure A.5: Quasi-optimal parameters $(\boldsymbol{\gamma}, \boldsymbol{\beta})$ found using the pyQuil INTERP method for the cyclic graph of 12 nodes for $p = 3, 4, 5$ and 10.

B

Runtime

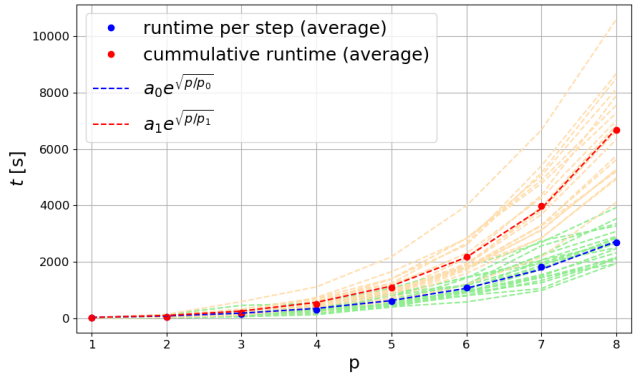


Figure B.1: Runtime (wall-clock time) when simulating the algorithm using the pyQuil INTERP method on 20 instances of 3-regular graph with 16 nodes. The fit parameters for the runtime per step are $a_0 = 2.4 \pm 0.6$, $p_0 = 6.1 \pm 0.4$. The fit parameters for the cummulative runtime are $a_1 = 1.4 \pm 0.2$ and $p_1 = 9.0 \pm 0.3$. The mean runtime per graph per step at $p = 8$ is around 2700 seconds, or 45 minutes. A run for one graph starting from $p = 1$ typically lasted 2 hours. The light green and navajo white dashed lines indicate the runtime per step and the cummulative time for one instance, respectively.

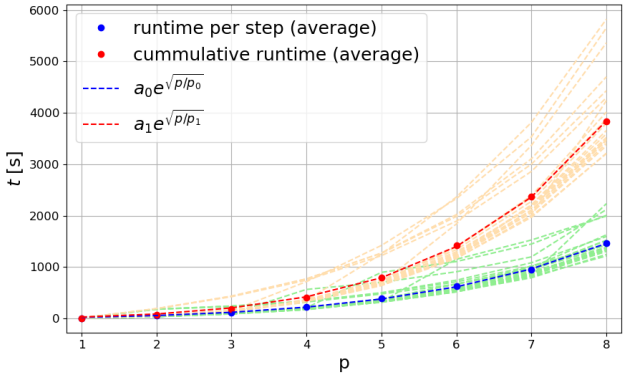


Figure B.2: Runtime (wall-clock time) when simulating the algorithm using the pyQuil INTERP method on 20 instances of weighted 3-regular graph with 16 nodes. The fit parameters for the runtime per step are $a_0 = 2.3 \pm 0.2$, $p_0 = 5.2 \pm 0.2$. The fit parameters for the cummulative runtime are $a_1 = 2.0 \pm 0.1$ and $p_1 = 7.1 \pm 0.1$. The light green and navajo white dashed lines indicate the runtime per step and the cummulative time for one instance, respectively. As can be seen, the parameters for weighted graphs typically took less time to calculate compared to unweighted graphs.

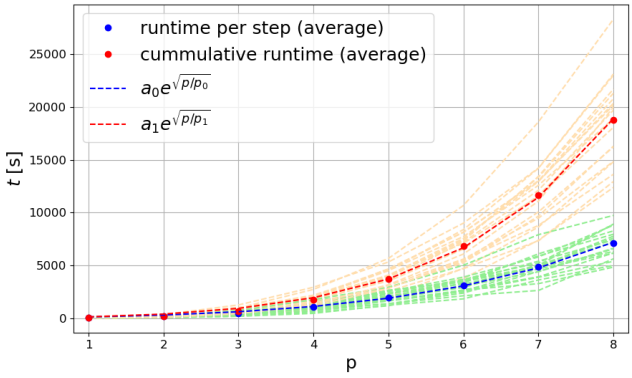


Figure B.3: Runtime (wall-clock time) when simulating the algorithm using the pyQuil INTERP method on 20 instances of ER graphs with 15 nodes and edge probability 0.5. The fit parameters for the runtime per step are $a_0 = 2.4 \pm 0.6$, $p_0 = 6.1 \pm 0.4$. The fit parameters for the cummulative runtime are $a_1 = 1.4 \pm 0.2$ and $p_1 = 9.0 \pm 0.3$. The mean runtime per graph per step at $p = 8$ is around 2700 seconds, or 45 minutes. A run for one graph starting from $p = 1$ typically lasted 2 hours. The light green and navajo white dashed lines indicate the runtime per step and the cummulative time for one instance, respectively.

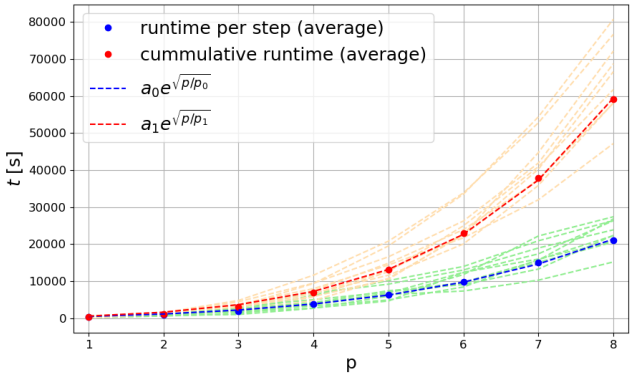
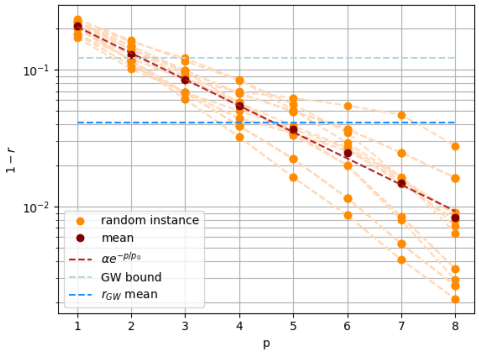


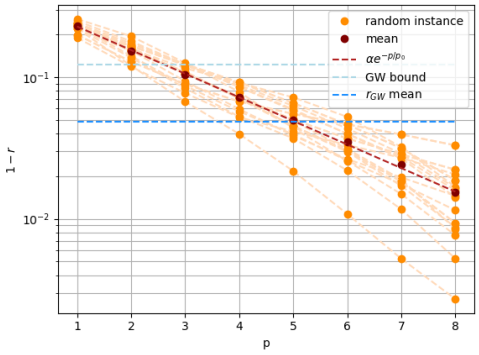
Figure B.4: Runtime (wall-clock time) when simulating the algorithm using the pyQuil INTERP method on 10 instances of weighted ER graphs with 15 nodes and edge probability 0.75. The fit parameters for the runtime per step are $a_0 = 60 \pm 7$, $p_0 = 4.3 \pm 0.2$. The fit parameters for the cummulative runtime are $a_1 = 44 \pm 5$ and $p_1 = 6.5 \pm 0.2$. The light green and navajo white dashed lines indicate the runtime per step and the cummulative time for one instance, respectively.

C

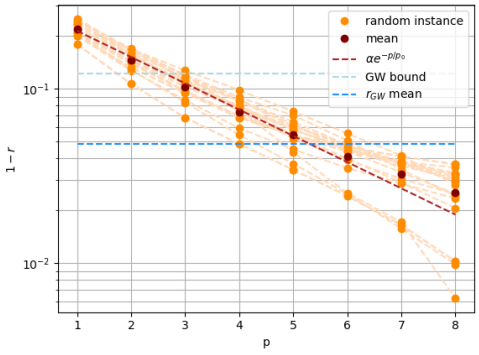
Fractional errors



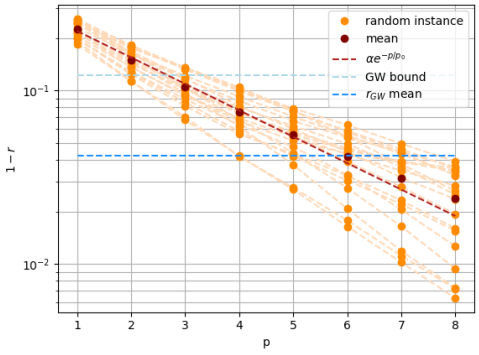
(a) 10 nodes. Fit parameters $\alpha = 0.322 \pm 0.004$ and $p_0 = 2.26 \pm 0.03$.



(b) 12 nodes. Fit parameters $\alpha = 0.335 \pm 0.002$ and $p_0 = 2.61 \pm 0.02$.

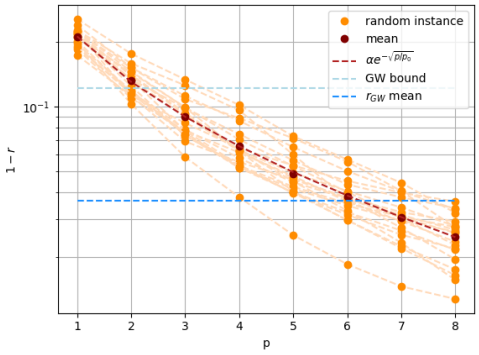


(c) 14 nodes. Fit parameters $\alpha = 0.30 \pm 0.01$ and $p_0 = 2.9 \pm 0.1$.

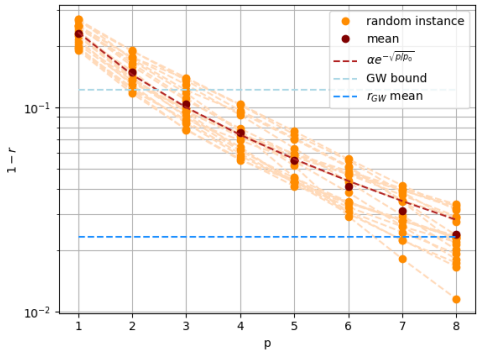


(d) 16 nodes. Fit parameters $\alpha = 0.313 \pm 0.009$ and $p_0 = 2.9 \pm 0.1$.

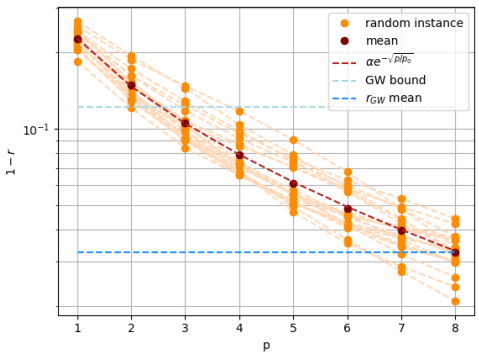
Figure C.1: The fractional error as a function of p for unweighted 3-regular graphs using INTERP. For each n 20 randomly generated instances were considered. A fit of the mean of the fractional error of those instances is included with model function $\alpha e^{-p/p_0}$ which suggests that the fractional error decays exponentially with p.



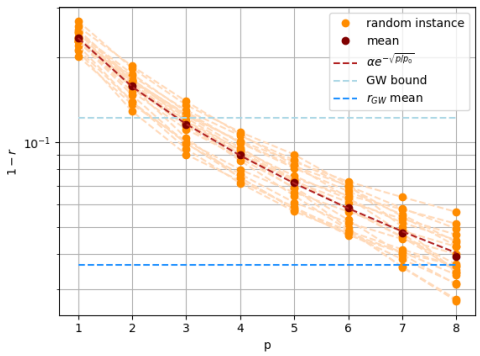
(a) 10 nodes. Fit parameters $\alpha = 0.688 \pm 0.006$ and $p_0 = 0.723 \pm 0.008$.



(b) 12 nodes. Fit parameters $\alpha = 0.740 \pm 0.03$ and $p_0 = 0.75 \pm 0.04$.

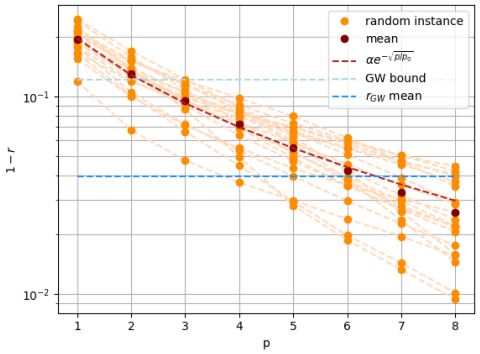


(c) 14 nodes. Fit parameters $\alpha = 0.654 \pm 0.006$ and $p_0 = 0.90 \pm 0.01$.

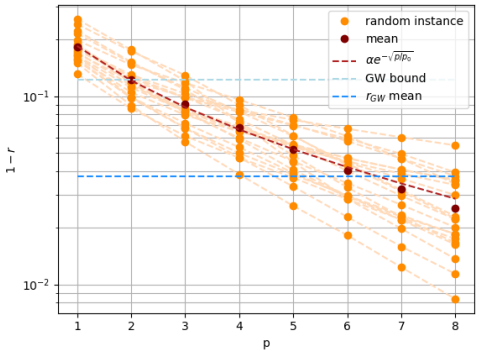


(d) 16 nodes. Fit parameters $\alpha = 0.615 \pm 0.005$ and $p_0 = 1.08 \pm 0.01$.

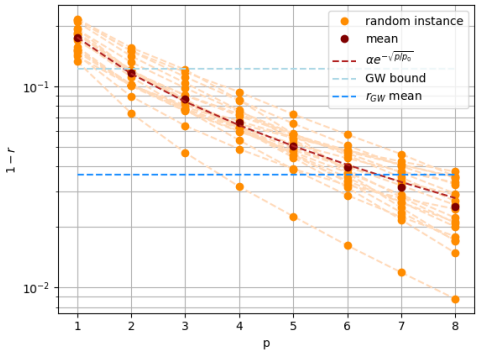
Figure C.2: The fractional error as a function of p for weighted 3-regular graphs using INTERP. For each n 20 randomly generated instances were considered. A fit of the mean of the fractional error of those instances is included with model function $\alpha e^{-\sqrt{p/p_0}}$ which suggests that the fractional error decays exponentially with the square root of p.



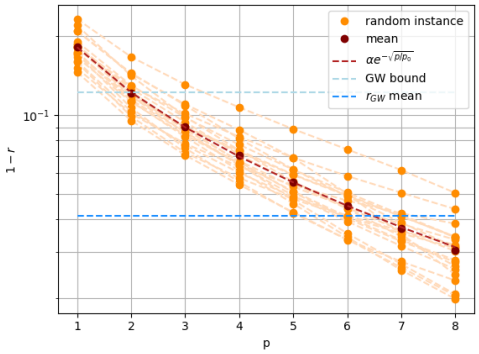
(a) 11 nodes. Fit parameters $\alpha = 0.55 \pm 0.02$ and $p_0 = 0.93 \pm 0.04$.



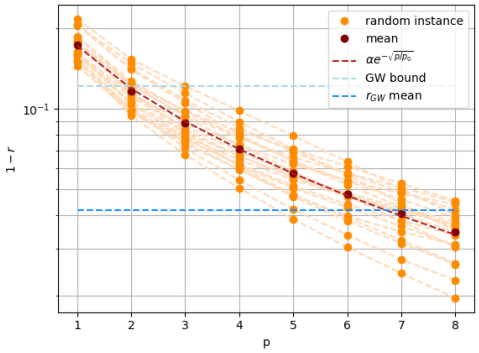
(b) 12 nodes. Fit parameters $\alpha = 0.51 \pm 0.01$ and $p_0 = 0.96 \pm 0.04$.



(c) 13 nodes. Fit parameters $\alpha = 0.48 \pm 0.01$ and $p_0 = 0.99 \pm 0.03$.

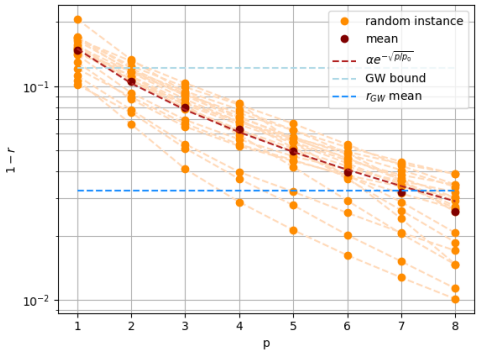


(d) 14 nodes. Fit parameters $\alpha = 0.474 \pm 0.004$ and $p_0 = 1.09 \pm 0.01$.

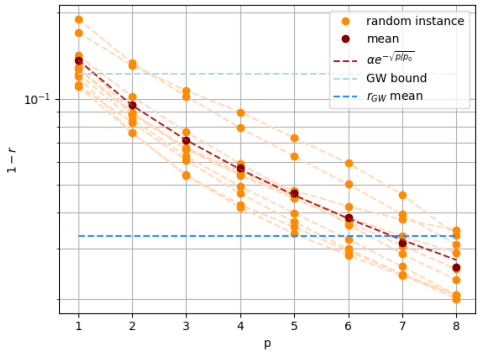


(e) 15 nodes. Fit parameters $\alpha = 0.420 \pm 0.006$ and $p_0 = 1.26 \pm 0.03$.

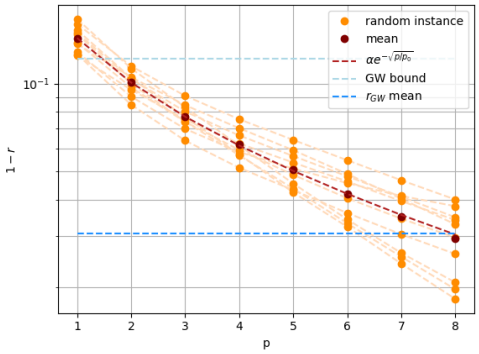
Figure C.3: The fractional error as a function of p for ER-0.5 graphs using INTERP. For each n 20 randomly generated instances were considered. A fit of the mean of the fractional error of those instances is included with model function $\alpha e^{-\sqrt{p/p_0}}$ which suggests that the fractional error decays exponentially with the square root of p.



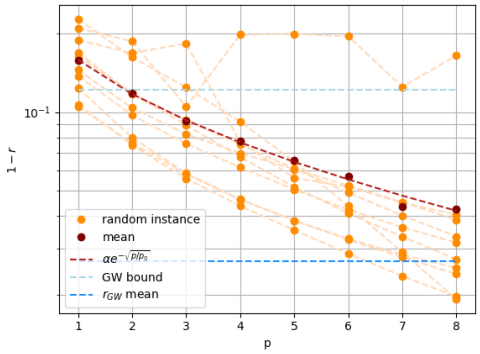
(a) 11 nodes. Fit parameters $\alpha = 0.37 \pm 0.01$ and $p_0 = 1.24 \pm 0.06$.



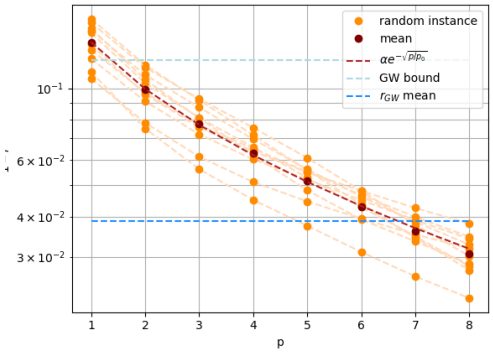
(b) 12 nodes. Fit parameters $\alpha = 0.312 \pm 0.002$ and $p_0 = 1.4 \pm 0.02$.



(c) 13 nodes. Fit parameters $\alpha = 0.317 \pm 0.002$ and $p_0 = 1.42 \pm 0.01$.



(d) 14 nodes. Fit parameters $\alpha = 0.327 \pm 0.008$ and $p_0 = 1.89 \pm 0.08$. There seems to be an anomalous results in one of the 14 nodal graphs.



(e) 14 nodes. Fit parameters $\alpha = 0.312 \pm 0.003$ and $p_0 = 1.53 \pm 0.02$.

Figure C.4: The fractional error as a function of p for ER-0.75 graphs using INTERP. For each n 10 randomly generated instances were considered. A fit of the mean of the fractional error of those instances is included with model function $\alpha e^{-\sqrt{p/p_0}}$ which suggests that the fractional error decays exponentially with the square root of p.

D

Software and Python packages

Python version 3.7.7 was used throughout this work. A list of the Python used packages and the corresponding versions is included in Table D.1

Package	Version
cvxpy	1.1.1
cvxgraphalgs	0.1.2
matplotlib	3.1.3
numpy	1.18.5
pandas	1.0.3
pyquil	2.19.0
quantum-grove	1.7.0
networkx	0.3.4
scipy	1.4.1

Table D.1: Versions of relevant Python packages used for implementation.

Moreover, the Rigetti Forest Software Development Kit (SDK) was used, which includes pyQuil, the Rigetti Quil Compiler (quilc), and the Quantum Virtual Machine (qvm). Forest SDK version 2.0 was used for this thesis.