Manuscript Click here to view linked References

A study of the performance of classical minimizers in the Quantum Approximate Optimization Algorithm

Mario Fernández-Pendás[a,b], Elías F. Combarro[c,d], Sofia Vallecorsa[d], José Ranilla[c], Ignacio F. Rúa[e]

[a]*Department of Science and Technology of Polymers, University of the Basque Country (UPV/EHU), Donostia (Spain)*
[b]*Donostia International Physics Center (DIPC), Donostia (Spain)*
[c]*Computer Science Department, University of Oviedo (Spain)*
[d]*Openlab, CERN, Geneva (Switzerland)*
[e]*Mathematics Department, University of Oviedo (Spain)*

Abstract

The Quantum Approximate Optimization Algorithm (QAOA) was proposed as a way of finding good, approximate solutions to hard combinatorial optimization problems. QAOA uses a hybrid approach. A parametrized quantum state is repeatedly prepared and measured on a quantum computer to estimate its average energy. Then, a classical optimizer, running in a classical computer, uses such information to decide on the new parameters that are then provided to the quantum computer. This process is iterated until some convergence criteria are met. Theoretically, almost all classical minimizers can be used in the hybrid scheme. However, their behaviour can vary greatly in both the quality of the final solution and the time they take to find it.

In this work, we study the performance of twelve different classical optimizers when used with QAOA to solve the maximum cut problem in graphs. We conduct a thorough set of tests on a quantum simulator both, with and without noise, and present results that show that some optimizers can be hundreds of times more efficient than others in some cases.

Keywords: Quantum Approximate Optimization Algorithm, Combinatorial Optimization, Max-Cut, Function minimization

1. Introduction

Quantum computing [1] is a computational paradigm in which subatomic particle properties such as superposition, entanglement and interference are exploited to obtain algorithms which offer speed-ups over classical algorithms. For instance, Shor's algorithm [2] is a quantum method that can factor an integer in time which is polynomial in the number of bits of the input while the best classical algorithm that we currently have is superpolynomial, and Grover's algorithm [3] and some quantum walks [4, 5] attain a quadratic speed-up in the black-box model over any possible classical algorithm for searching an element in an unsorted list.

A fruitful topic of research has been the use of quantum computing for solving combinatorial optimization problems. One of the most popular approaches has been that of quantum adiabatic computing [6, 7]. In this model, a quantum system is initialized to a certain, well-known state and then it is made to evolve following a slowly changing Hamiltonian until it reaches the ground state of another Hamiltonian that codes the solution to our problem. Although this model is equivalent to the paradigm of quantum circuits and it is, thus, universal, its most popular incarnation has been through the use of the restricted version called quantum annealing [8], for instance in D-Wave's quantum computers [9].

The Quantum Approximate Optimization Algorithm (QAOA) [10] can be seen as a discretization of quantum adiabatic computing in which a target Hamiltonian and a mixer Hamiltonian are used in alternate steps to drive the evolution of the system to a (ground) state that encodes a solution to the optimization problem under study. Since its introduction, the properties of QAOA have been (and still are) the subject of intense study [11, 12, 13, 14, 15, 16, 17] and it has been applied to several different optimization problems [18, 19, 20, 21].

Email addresses: `mfernandez@dipc.org` (Mario Fernández-Pendás), `efernandezca@uniovi.es` (Elías F. Combarro), `Sofia.Vallecorsa@cern.ch` (Sofia Vallecorsa), `ranilla@uniovi.es` (José Ranilla), `rua@uniovi.es` (Ignacio F. Rúa)

QAOA follows a hybrid classical-quantum scheme. The classical computer is used to determine the optimal values in a parametrized quantum state that has a high overlap with (a good approximation to) the solution of our optimization problem. Then, a quantum computer is used to prepare that quantum state with the optimal parameters found by the classical algorithm. When that state is measured, a good solution is found with high probability.

Theoretically, the optimal parameters for the quantum state can be obtained efficiently with a classical algorithm for some problems [10]. However, although the exact method has asymptotical polynomial complexity, in practice its running time can be rather large. Thus, an iterative approach is usually adopted instead. The classical computer starts with a guess of the optimal parameters and uses the quantum computer to evaluate the expected value of the quantum state for those parameters. Then, a classical minimizer or maximizer is used to obtain the optimal values. Throughout this process, the quantum computer is used as a black-box capable of providing (estimations of) the expected values of the quantum state for different values of the parameters.

Although QAOA has been the subject of many research papers, not so much attention has been paid to the influence of the choice of classical optimizer in the efficacy and efficiency of the method and, to a prospective user of the algorithm it can be difficult to select the most appropriate minimizer among the many existing ones. In this work, we focus exactly on that problem and we study the performance of twelve different classical optimizers when used together with QAOA to solve the maximum cut problem in graphs, with the aim of providing a guide to help in choosing the best minimizer to use with QAOA. We conduct a series of numerical experiments in a quantum simulator, both with and without noise, and analyse the results in terms of quality of the solution and of the computation time needed to achieve it. This shows that some optimizers can be hundreds of times faster than others while attaining approximations to the solution of similar quality, so the choice of the classical optimizer should be a key factor when using QAOA. Partial studies of the performance of optimizers for QAOA have been conducted before, but only with a fraction of the twelve methods that we examine in this work (for instance, in [22] and [23] three optimizers are studied, while five are considered in [24]).

The rest of the paper is organized as follows. In Section 2 we present the mathematical formulation of QAOA and we show how approximate solutions to NP-hard problems such as Max-Cut can be found with it and in Section 3 we derive the analytical expression of the expected energy for some of the cases we study in this paper. In Section 4, we introduce the classical optimizers that we have used in under study and present some of their main properties. In Section 5 we describe the settings of our experiments and present and analyse the results we have obtained. Finally, in Section 6 we raise some conclusions and propose ideas for future work.

2. QAOA and the Max-Cut problem

The objective of QAOA is to find a good approximation of the ground state of a given Hamiltonian C_H (below, we will show how we can use this Hamiltonian to encode instances of NP-hard optimization problems). In the general case, this problem is very difficult to solve even when C_H is diagonal in the computational basis so we start with another Hamiltonian I_H whose ground state $|\phi\rangle$ is easy to find and prepare (usually, $|\phi\rangle = |+\rangle^{\otimes n}$). Then, we consider the parametrized quantum state

$$|\beta, \gamma\rangle = e^{-i\beta_p I_H} e^{-i\gamma_p C_H} \dots e^{-i\beta_2 I_H} e^{-i\gamma_2 C_H} e^{-i\beta_1 I_H} e^{-i\gamma_1 C_H} |\phi\rangle,$$

where β and γ are real number p-tuples which are called angles, and $p \geq 1$ is a parameter. Note that this can be seen as a discretization or Trotterization [25] of the evolution of $|\phi\rangle$ under the time-depending Hamiltonian that is used in quantum adiabatic computing which is given, for $t \in [0, T]$ (where T is the total time of the evolution), by

$$H(t) = \left(1 - \frac{t}{T}\right) I_H + \frac{t}{T} C_H.$$

In QAOA, a classical computer is used to find the values β and γ that minimize the expected value

$$F(\beta, \gamma) = \langle \beta, \gamma | C_H | \beta, \gamma \rangle.$$

Then, the state with optimal angles is prepared in the quantum computer. When measured, the resulting state is a good approximation to the combinatorial problem that is coded via C_H. Obviously, the bigger p, the better the solution that we obtain and, in fact, the sequence of the optimal quantum states can be seen [10] as converging to the optimal solution to the problem when $p \to \infty$.

For some cases, it has been shown [10] that the optimal angles β and γ can be found efficiently (i.e. with an algorithm that takes time polynomial in the dimension of the Hilbert space) with a classical computer. However, this polynomial can be of very high degree and, in practice, the usual approach is to use a classical function optimizer to obtain good values of β and γ with the quantum computer acting as a black-box device that returns (estimations of) $F(\beta, \gamma)$. That is, the classical optimizer uses the quantum computer to (stochastically) evaluate and minimize F. Once that the stopping criteria of the classical optimizer are met, the quantum computer is used to prepare a number of copies of $|\beta, \gamma\rangle$ with the optimal angles. These copies of the state are measured and the best solution found this way is returned. This is summarized in Algorithm 1.

Algorithm 1 Quantum Approximate Optimization Algorithm

```
1:  INPUT: $p$, $C_H$, K
2:  OUTPUT: An approximate solution $z$
3:  while stopping criteria not met do
4:      Choose $p$ real values for $\beta$ and for $\gamma$ (classical computer)
5:      Prepare state $|\beta, \gamma\rangle$ (quantum computer)
6:      Estimate $\langle\beta, \gamma|C_H|\beta, \gamma\rangle$ (classical and quantum computer)
7:  end while
8:  for i:=1 to K do
9:      Prepare $|\beta, \gamma\rangle$ (quantum computer)
10:     Measure $|\beta, \gamma\rangle$ to obtain $|z\rangle$ (quantum computer)
11:     Compute $F(z) = \langle z|C_H|z\rangle$ (classical computer)
12: end for
13: Return $z$ with the lowest value of $F(z)$
```

One of the problems that have been more extensively used with QAOA is the maximum cut (or Max-Cut) problem in graphs. This is is one of the most studied combinatorial optimization problems and has applications in statistical physics and circuit layout design, among other fields [26, 27]. It has been shown to be NP-hard [28] and it remains NP-hard even when the graphs are constrained to have maximum degree 3 [29].

The Max-Cut problem can be stated as, given a graph $G = (V, E)$, find a partition of the vertices V in two sets such that the number of edges $e \in E$ with extremes in different sets of the partition is the maximum possible. This can be easily reformulated as the problem of finding the ground state of a Hamiltonian in the following way. We consider a Hilbert space of dimension $n = |V|$. We identify each state $|z\rangle$ of the computational basis of the Hilbert space with a partition of V in a straightforward way: if $z_i = 0$, then vertex i belongs to the first set of the partition and if $z_i = 1$, it belongs to the second. Then, an edge $e = \{i, j\} \in E$ has extremes in different sides of the partition given by z if and only if

$$\langle z|Z_i \otimes Z_j|z\rangle = -1,$$

where $Z_i \otimes Z_j$ is the tensor product of the Z Pauli gates on qubits i and j, respectively.

It is easy to see, then, that the Max-Cut problem can be alternatively stated as finding a ground state of the Hamiltonian

$$C_H = \sum_{\{i,j\} \in E} Z_i \otimes Z_j. \tag{1}$$

3. Analytic study of the first level QAOA for Max-Cut

In this section we obtain an explicit expression for the energy expectation related to the cost Hamiltonian C_H, for QAOA of depth level $p = 1$. We start by fixing some notation.

Definition 1. Let n and t be natural numbers, and let $H \in \mathcal{M}_{t\times n}(\mathbb{F}_2)$ be the incidence matrix of an undirected graph with no loops. This means that the graph consists of n vertices and t edges $E_\nu = \{i_\nu, j_\nu\}$ (with $i_\nu \neq j_\nu$), with $H_{\nu i_\nu} = H_{\nu j_\nu} = 1$, and zero elsewhere. We define the cost Hamiltonian

$$C_H = \sum_{\nu=1}^{t} C_\nu \ , \text{ where } C_\nu = Z_{i_\nu} \otimes Z_{j_\nu}.$$

For simplicity, since $p = 1$, we identify $\beta = \beta_1, \gamma = \gamma_1 \in [0, 2\pi]$. Let $U(C_H, \gamma) = \exp(-i\gamma C_H)$, $U(B, \beta) = \exp\left(-i\beta \sum_{i=1}^{n+r} X_i\right)$, $|\phi\rangle = |+\rangle^{\otimes n}$, so that the final state of QAOA of depth level $p = 1$ is

$$|\gamma, \beta\rangle = U(B, \beta)U(C_H, \gamma)|\phi\rangle$$

and the expected value of C_H in the final state is

$$F_1(\gamma, \beta) = \langle\gamma, \beta|C_H|\gamma, \beta\rangle.$$

In order to find an explicit expression of such a final state, let us introduce:

- For all $\nu = 1, \dots, t$: $I_\nu = \{k \in \{1, \dots, n\} \mid \mathbf{H}_{\nu k} = 1\}$.
- For all $l \in \mathbb{N}$, and $E \subseteq \{1, \dots, l\}$: $\chi^l_E = \{x \in \mathbb{F}_2^l \mid x_i = 0 \text{ , if } i \notin E\}$.
- For all $\nu \in \{1, \dots, t\}$, and for all $b \in \chi^n_{I_\nu}$: $\Theta_{\nu,b} = \{\mu \in \{1, \dots, t\} \mid \mathbf{H}_\mu \cdot b^t = 1\}$, where $\mathbf{H}_\mu$ denotes the $\mu-$th row of the matrix $\mathbf{H}$.
- For all $\nu \in \{1, \dots, t\}$: $\Lambda_\nu = \left\{(b, a) \in \chi^n_{I_\nu} \times \chi^t_{\Theta_{\nu,b}} \mid a \cdot \mathbf{H} = H_\nu\right\}$.
- $wt_H(b)$ denotes the Hamming weight of the binary array b.

Let us show specific expressions for those sets in our particular case. For all $\nu = 1, \dots, t$, the edge $E_\nu = \{i_\nu, j_\nu\}$ yields

$$I_\nu = \{i_\nu, j_\nu\}.$$

Consequently, the set $\chi^n_{I_\nu}$ consists of those binary arrays of length n whose ones are contained in positions i_ν or j_ν. Therefore, if we let 0^n be the length n zero vector, and we denote by e_k the binary array of length n with a one in the $k-$th position and zeroes elsewhere, we have that

$$\chi^n_{I_\nu} = \{0^n \ , \ e_{i_\nu} \ , \ e_{j_\nu} \ , \ e_{i_\nu} \oplus e_{j_\nu}\},$$

where $\oplus$ denotes the XOR operator.

Now, for all $\nu \in \{1, \dots, t\}$, and for all $b \in \chi^n_{I_\nu}$, we have:

- When $b = 0^n$, the product $H_\mu \cdot b^t$ is always zero, so that $\Theta_{\nu,0^n} = \emptyset$. Consequently, $\chi^t_{\Theta_{\nu,0^n}} = \{0^t\}$, $0^t \cdot \mathbf{H} = 0^n \neq H_\nu$, and so the pair $(0^n, 0^t) \notin \Lambda_\nu$.
- When $b = e_{i_\nu}$, the product $H_\mu \cdot b^t$ is equal to one exactly when $H_{\mu i_\nu} = 1$, i.e., when the vertex i_ν is an endpoint of the edge E_μ, so that

  $$\Theta_{\nu,e_{i_\nu}} = \{\mu \mid \text{the vertex } i_\nu \text{ is incident with the edge } E_\mu\}.$$

  Consequently, $\chi^t_{\Theta_{\nu,e_{i_\nu}}}$ consists of arrays a with ones in positions μ such that $E_\mu = \{i_\nu, j_\mu\}$ (in particular, $e_\nu \in \chi^t_{\Theta_{\nu,e_{i_\nu}}}$). Observe that for all $\mu \neq \nu$, we have that $j_\mu \neq j_\nu$. Therefore, if a contains a one in a position $\mu \neq \nu$, we have that $a \cdot \mathbf{H}$ contains a one in j_μ, and so $a \cdot \mathbf{H} \neq H_\nu$. This means that only $(e_{i_\nu}, e_\nu) \in \Lambda_\nu$.
- Analogous properties hold when $b = e_{j_\nu}$.

- When $b = e_{i_\nu} \oplus e_{j_\nu}$, the set $\Theta_{\nu, e_{i_\nu} \oplus e_{j_\nu}}$ consists of those μ such that the vertices i_ν, j_ν are incident with the edge E_μ, but not both at the same time, i.e., $\Theta_{\nu, e_{i_\nu} \oplus e_{j_\nu}} = \Theta_{\nu, e_{i_\nu}} \triangle \Theta_{\nu, e_{j_\nu}}$, where $\triangle$ denotes the symmetric difference of sets. Consequently, $\chi^t_{\Theta_{\nu, e_{i_\nu} \oplus e_{j_\nu}}}$ consists of arrays a with ones in positions $\mu \neq \nu$ such that the edge E_μ contains either the vertex i_ν or the vertex j_ν.

  Now, assume that a contains a one in the $\mu-$th position, i.e, that $E_\mu = \{i_\nu, j_\mu\}$ or $E_\mu = \{i_\mu, j_\nu\}$. Then, for $a \cdot \mathbf{H}$ to be equal to H_ν, a must contain a one also in a position μ' such that the edge $E_{\mu'}$ is of the form $\{j_\mu, j_\nu\}$, in the first case, or of the form $\{i_\mu, i_\nu\}$, in the second one. In both cases, a triangle in the graph is formed by the edges E_ν, E_μ, and $E_{\mu'}$. So, a has ones in positions forming triangles of the graph containing the edge E_ν. Moreover, such number of triangles must be odd, in order for $a \cdot \mathbf{H}$ to be equal to H_ν (otherwise, it is equal to the zero array). Let us denote by T_ν the set of such a.

Summarizing, for all $\nu \in \{1, \ldots, t\}$, $\Lambda_\nu = \{(e_{i_\nu}, e_\nu), (e_{j_\nu}, e_\nu)\} \cup \{e_{i_\nu} \oplus e_{j_\nu}\} \times T_\nu$. We shall use the following auxiliary lemma.

Lemma 1. *With the previous notation, for all $\nu = 1, \ldots, t$,*

1. $$U(B,\beta)^\dagger \left(\bigotimes_{k \in I_\nu} Z_k \right) U(B,\beta) = \sum_{b \in \chi^n_{I_\nu}} \cos(2\beta)^{2 - wt_H(b)} \sin(2\beta)^{wt_H(b)} \bigotimes_{k \in I_\nu} Z_k^{1-b_k} Y_k^{b_k}$$

2. *For all $b \in \chi^n_{I_\nu}$,*

$$\begin{aligned} &U(C_H,\gamma)^\dagger \left(\bigotimes_{k \in I_\nu} Z_k^{1-b_k} Y_k^{b_k} \right) U(C_H,\gamma) \\ &\quad = \sum_{a \in \chi^{n+r}_{\Theta_{\nu,b}}} \prod_{\mu \in \Theta_{\nu,b}} \left(\cos(2\gamma)^{1-a_\mu} (-i \sin(2\gamma))^{a_\mu} \right) \left(\bigotimes_{l=1}^n Z_l^{(a \cdot \mathbf{H})_l} \right) \left(\bigotimes_{k \in I_\nu} Z_k^{1-b_k} Y_k^{b_k} \right). \end{aligned}$$

Proof. 1. Since $U(B,\beta) = \exp\left(-i\beta_j \sum_{l=1}^n X_l\right) = \prod_{l=1}^n \exp(-i\beta_l X_j) = \prod_{l=1}^n (\cos(\beta) I + i \sin(\beta) X_l)$, we have

$$U(B,\beta)^\dagger \left(\bigotimes_{k \in I_\nu} Z_k \right) U(B,\beta) = \prod_{l=1}^n (\cos(\beta) I + i \sin(\beta) X_l) \left(\bigotimes_{k \in I_\nu} Z_k \right) \prod_{l=1}^n (\cos(-\beta) I + i \sin(-\beta) X_l).$$

When $l \notin I_\nu$, the terms $(\cos(\beta) I + i \sin(\beta) X_l)$, $\left(\bigotimes_{k \in I_\nu} Z_k\right)$ commute, so we have

$$\begin{aligned} &\bigotimes_{k \in I_\nu} \left((\cos(\beta) I + i \sin(\beta) X_k) Z_k (\cos(-\beta) I + i \sin(-\beta) X_k) \right) \\ &\quad = \bigotimes_{k \in I_\nu} \left((\cos(\beta) Z_k + \sin(\beta) Y_k)(\cos(\beta) I - i \sin(\beta) X_k) \right) \\ &\quad = \bigotimes_{k \in I_\nu} \left((\cos(\beta) Z_k + \sin(\beta) Y_k)(\cos(\beta) I - i \sin(\beta) X_k) \right) \\ &\quad = \bigotimes_{k \in I_\nu} \left((\cos^2(\beta) - \sin^2(\beta)) Z_k + 2 \sin(\beta) \cos(\beta) Y_k \right) \\ &\quad = \bigotimes_{k \in I_\nu} (\cos(2\beta) Z_k + \sin(2\beta) Y_k) = \sum_{b \in \chi^n_{I_\nu}} \bigotimes_{k \in I_\nu} (\cos(2\beta) Z_k)^{1-b_k} (\sin(2\beta) Y_k)^{b_k} \\ &\quad = \sum_{b \in \chi^n_{I_\nu}} \cos(2\beta)^{2 - wt_H(b)} \sin(2\beta)^{wt_H(b)} \bigotimes_{k \in I_\nu} Z_k^{1-b_k} Y_k^{b_k} \end{aligned}$$

after expansion of the tensor product as a sum of terms.

2. Let us inspect the conjugation of $\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}$ by the $\mu-$th factor of the operator

$$U(C_H,\gamma_j) = \exp(-i\gamma_j C_H) = \exp\left(i\gamma_j \sum_{\mu=1}^{t} \bigotimes_{l\in I_\mu} Z_l\right) = \prod_{\mu=1}^{t} \exp\left(i\gamma_j \bigotimes_{l\in I_\mu} Z_l\right).$$

Since $\exp\left(i\gamma_j \bigotimes_{l\in I_\mu} Z_l\right) = \cos(\gamma_j)I + i\sin(\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)$, we have

$$\begin{aligned}
&\exp\left(i\gamma_j \bigotimes_{l\in I_\mu} Z_l\right)^\dagger \left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right) \exp\left(i\gamma_j \bigotimes_{l\in I_\mu} Z_l\right)\\
&\quad= \left(\cos(-\gamma_j)I + i\sin(-\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\right)\cdot\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\left(\cos(\gamma_j)I + i\sin(\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\right)\\
&\quad= \cos^2(-\gamma_j)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right) + \sin^2(-\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\left(\bigotimes_{l\in I_\mu} Z_l\right)\\
&\quad- i\sin(-\gamma_j)\cos(-\gamma_j)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\left(\bigotimes_{l\in I_\mu} Z_l\right) + i\sin(-\gamma_j)\cos(-\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right).
\end{aligned}$$

Since Z_k and Y_k anticommute in positions $k\in I_\mu$ such that $b_k = 1$, and commute otherwise, we have

$$\begin{aligned}
&= \cos^2(-\gamma_j)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right) + (-1)^{\mathbf{H}_\mu\cdot b^t}\sin^2(-\gamma_j)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\\
&- i(-1)\sin(-\gamma_j)\cos(-\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right) + i\sin(-\gamma_j)\cos(-\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\\
&= \left(\cos(-2\gamma_j)I + i\sin(-2\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\right)^{\mathbf{H}_\mu\cdot b^t}\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)
\end{aligned}$$

because of elementary trigonometric relations. Observe that the first factor contains only Z matrices, so it commutes with the remaining factors of the operator $U(C_H,\gamma_j)$. Therefore,

$$\begin{aligned}
U(C_H,\gamma)^\dagger\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)U(C_H,\gamma) &= \prod_{\mu=1}^{t}\exp\left(i\gamma_j \bigotimes_{l\in I_\mu} Z_l\right)^\dagger\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\cdot\prod_{\mu=1}^{t}\exp\left(i\gamma_j \bigotimes_{l\in I_\mu} Z_l\right)\\
&= \left(\prod_{\mu=1}^{t}\left(\left(\cos(-2\gamma_j)I + i\sin(-2\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\right)^{\mathbf{H}_\mu\cdot b^t}\right)\right)\cdot\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\\
&= \left(\prod_{\mu\in\Theta_{\nu,b}}\left(\cos(-2\gamma_j)I + i\sin(-2\gamma_j)\left(\bigotimes_{l\in I_\mu} Z_l\right)\right)\right)\cdot\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\\
&= \sum_{a\in\chi^t_{\Theta_{\nu,b}}}\prod_{\mu\in\Theta_{\nu,b}}\left(\cos(-2\gamma)^{1-a_\mu}(i\sin(-2\gamma))^{a_\mu}\left(\bigotimes_{k\in I_\mu} Z_k\right)^{a_\mu}\right)\cdot\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)\\
&= \sum_{a\in\chi^t_{\Theta_{\nu,b}}}\prod_{\mu\in\Theta_{\nu,b}}\left(\cos(-2\gamma)^{1-a_\mu}(i\sin(-2\gamma))^{a_\mu}\right)\cdot\prod_{\mu=1}^{t}\left(\bigotimes_{k\in I_\mu} Z_k\right)^{a_\mu}\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k} Y_k^{b_k}\right)
\end{aligned}$$

after expansion of the product as a sum of terms, and because $a_\mu = 0$ when $\mu \notin \Theta_{\nu,b}$, i.e., when $a \in \chi^t_{\Theta_{\nu,b}}$. Now, for all $l = 1, \ldots, t$, the Pauli matrix Z_l appears in the product $\prod_{\mu=1}^t \left(\bigotimes_{k\in I_\mu} Z_k\right)^{a_\mu}$ if and only if $l \in I_\mu$ and $a_\mu = 1$, i.e., if and only if $a_\mu \cdot \mathbf{H}_{\mu,l} = 1$. Because $Z_l^2 = I$, and the number this happens is $(a\cdot\mathbf{H})_l$ mod 2, we conclude that $\prod_{\mu=1}^t \left(\bigotimes_{k\in I_\mu} Z_k\right)^{a_\mu} = \bigotimes_{l=1}^n Z_l^{(a\cdot\mathbf{H})_l}$.

□

Theorem 1. *With the previous notation, the cost expectation related to the cost Hamiltonian C_H, for QAOA of depth level $p = 1$, is*

$$\begin{aligned}F_1(\gamma,\beta) =& \sum_{\nu=1}^t \cos(2\beta)\sin(2\beta)\left(\cos(2\gamma)^{\#E(i_\nu)} + \cos(2\gamma)^{\#E(j_\nu)}\right)\sin(2\gamma)\\ &+\sum_{\nu=1}^t\sum_{a\in T_\nu}(-i\sin(2\beta))^2\cos(2\gamma)^{\#E(i_\nu)+\#E(j_\nu)-2-wt_H(a)}(i\sin(2\gamma))^{\#wt_H(a)}\end{aligned}$$

Proof. Since $F_1(\gamma,\beta) = \langle\gamma,\beta|C_H|\gamma,\beta\rangle = \langle\phi|U(C_H,\gamma)^\dagger U(B,\beta)^\dagger C_H U(B,\beta)U(C_H,\gamma)|\phi\rangle$, we shall begin by inspecting the inner part of the expression. Because of Lemma 1,

$$\begin{aligned}F_1(\gamma,\beta) &= \left\langle \phi \left| U(C_H,\gamma)^\dagger U(B,\beta)^\dagger \left(\sum_{\nu=1}^t \bigotimes_{k\in I_\nu} Z_k\right) U(B,\beta)U(C_H,\gamma)\right| \phi\right\rangle\\
&= \sum_{\nu=1}^t \left\langle \phi \left| U(C_H,\gamma)^\dagger \left(U(B,\beta)^\dagger \left(\bigotimes_{k\in I_\nu} Z_k\right) U(B,\beta)\right) U(C_H,\gamma)\right| \phi\right\rangle\\
&= -\sum_{\nu=1}^{n+r} \langle \phi \left| U(C_H,\gamma)^\dagger \right. \cdot \left(\sum_{b\in\chi^n_{I_\nu}} \cos(2\beta)^{2-wt_H(b)}\sin(2\beta)^{wt_H(b)} \bigotimes_{k\in I_\nu} Z_k^{1-b_k}Y_k^{b_k}\right) U(C_H,\gamma)\left| \phi\right\rangle\\
&= \sum_{\nu=1}^t \sum_{b\in\chi^n_{I_\nu}} \cos(2\beta)^{2-wt_H(b)}\sin(2\beta)^{wt_H(b)} \cdot \left\langle \phi \left| U(C_H,\gamma)^\dagger \left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k}Y_k^{b_k}\right) U(C_H,\gamma)\right| \phi\right\rangle\\
&= \sum_{\nu=1}^t \sum_{b\in\chi^n_{I_\nu}} \cos(2\beta)^{2-wt_H(b)}\sin(2\beta)^{wt_H(b)} \cdot\\
&\quad\cdot \left\langle \phi \left| \sum_{a\in\chi^t_{\Theta_{\nu,b}}} \prod_{\mu\in\Theta_{\nu,b}} \left(\cos(2\gamma)^{1-a_\mu}(i\sin(2\gamma))^{a_\mu}\right) \left(\bigotimes_{l=1}^n Z_l^{(a\cdot\mathbf{H})_l}\right)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k}Y_k^{b_k}\right)\right| \phi\right\rangle\\
&= \sum_{\nu=1}^t \sum_{b\in\chi^n_{I_\nu}} \sum_{a\in\chi^t_{\Theta_{\nu,b}}} \cos(2\beta)^{2-wt_H(b)}\sin(2\beta)^{wt_H(b)} \cdot \prod_{\mu\in\Theta_{\nu,b}} \left(\cos(2\gamma)^{1-a_\mu}(i\sin(2\gamma))^{a_\mu}\right)\\
&\quad\cdot \left\langle \phi \left| \left(\bigotimes_{l=1}^n Z_l^{(a\cdot\mathbf{H})_l}\right)\left(\bigotimes_{k\in I_\nu} Z_k^{1-b_k}Y_k^{b_k}\right)\right| \phi\right\rangle.\end{aligned}$$

When $l \in I_\nu$, the $l-$th term in the tensor product is $Z_l^{(a\cdot\mathbf{H})_l+1-b_l}Y_l^{b_l}$, whereas when $l \notin I_\nu$, it is $Z_l^{(a\cdot\mathbf{H})_l}$. Since $\langle +|X_l|+\rangle = \langle +|I_l|+\rangle = 1$, and $\langle +|Z_l|+\rangle = \langle +|Y_l|+\rangle = 0$, the only non-zero terms in the previous sum are those with all tensor factors indexed by $l \in I_\nu$, and $(a\cdot\mathbf{H})_l = 1$, or by $l \notin I_\nu$, $(a\cdot\mathbf{H})_l = 0$. So, for all $l = 1, \ldots, n$, we must have $(a\cdot\mathbf{H})_l = H_l$. Finally, observe that a product $Z_lY_l = -iX_l$ occurs only when $l \in I_\nu, (a\cdot\mathbf{H})_l = 1$, and $b_l = 1$, introducing a factor $-i$ in the addition term. Since this happens $wt_H(b) = 1$ times, we obtain:

$$F_1(\gamma,\beta) = \sum_{\nu=1}^t \sum_{(b,a)\in\Lambda_\nu} \cos(2\beta)^{2-wt_H(b)}(-i\sin(2\beta))^{wt_H(b)} \cdot \prod_{\mu\in\Theta_{\nu,b}} \left(\cos(2\gamma)\right)^{1-a_\mu}\left(i\sin(2\gamma)\right)^{a_\mu}$$

Now, for all $\nu = 1, \ldots, t$ we have the following possibilities for $(b, a) \in \Lambda_\nu$:

- (e_{i_ν}, e_ν) : in this case the corresponding summand in $F_1(\gamma, \beta)$ is

$$\cos(2\beta)(-i\sin(2\beta))\cos(2\gamma)^{\#E(i_\nu)}(i\sin(2\gamma)) = \cos(2\beta)\sin(2\beta)\cos(2\gamma)^{\#E(i_\nu)}\sin(2\gamma),$$

  where $E(i_\nu)$ denotes the set of edges containing the vertex i_ν.

- (e_{j_ν}, e_ν) : similarly, the corresponding summand in $F_1(\gamma, \beta)$ is

$$\cos(2\beta)\sin(2\beta)\cos(2\gamma)^{\#E(j_\nu)}\sin(2\gamma).$$

- $(e_{i_\nu} + e_{j_\nu}, a)$, where $a \in T_\nu$: the summand in $F_1(\gamma, \beta)$ is

$$(-i\sin(2\beta))^2\cos(2\gamma)^{\#E(i_\nu)+\#E(j_\nu)-2-wt_H(a)}(i\sin(2\gamma))^{\#wt_H(a)}.$$

□

4. Classical optimizers

In principle, any optimizer that does not need the explicit expression of the function to minimize (but only uses it as black-box to evaluate its values at some points) can be used in conjunction with QAOA. In this work, we have focused on those optimizers that are included in version 0.7.3 of the Aqua module [30] of IBM's Qiskit quantum programming language [31]. This is, probably, the most popular quantum programming language nowadays and the Aqua module natively implements QAOA. For this reason, for our experiments we have selected this implementation together with the twelve optimizers that Aqua provides.

The methods described below are used to solve optimization problems of the form[1]

$$\begin{array}{lcr} \min\limits_{x \in D} & & f(x) \\ \text{s.t.} & b_i(x) \leq 0, & i = 1, \ldots, m \\ & c_j(x) = 0 & j = 1, \ldots, n, \end{array} \tag{2}$$

where $D \subseteq \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function. We call f the objective function, and b_i and c_i the constraints. If $m = n = 0$, the problem is said to be unconstrained.

Classical optimizers can be classified in two main families: gradient-based and gradient-free methods. The former require the computation of the gradient of the objective function while the latter do not. We use this classification to explain, in the following paragraphs, the details of the algorithms. Unless noted otherwise, all the optimization methods described below use the implementation included in the *optimize* library of SciPy [32].

4.1. Gradient-based methods

Gradient-based methods use search directions defined by the gradients of the objective function f in (2) at certain points in D. For some given problems, the calculation of the required derivatives can become computationally demanding. On the other hand, gradient-based methods are generally preferred when the parameter space D becomes large. Thus, in some situations, they are the method of choice.

Adam [33] is an stochastic gradient descent algorithm [34]. It is an adaptive learning rate method. It relies on adaptive estimates of first and second moments of the gradients which accelerate the stochastic gradient descent in the relevant direction, as well as help dampening oscillations. Adam algorithm requires little memory and it is invariant to diagonal rescaling of the gradients. Furthermore, it is appropriate for non-stationary objective functions, and noisy and/or sparse gradients. There is also a variant of Adam called AMSGRAD [35] which uses a long-term memory of past gradients and, thereby, improves convergence properties. Another gradient descent optimization method which uses a momentum term is the Analytic Quantum Gradient Descent (AQGD)

[1]A maximization problem can be treated by negating the function f.

optimizer. The gradients are computed analytically for parameterized quantum gates (see [36, 37] for further details) using the quantum circuit when evaluating the objective function. The Simultaneous Perturbation Stochastic Approximation (SPSA) optimizer [38] is an algorithm for optimizing systems with multiple unknown parameters. SPSA is also a gradient-descent method. It does not involve momentum terms, but it provides a level of accuracy in the optimization of the objective function comparable with other gradient approximations such as finite-differences. Its main feature is the gradient approximation, which requires only two measurements of the objective function, regardless of the dimension of the optimization problem. Further details can be found in [39]. Both Adam and its variant AMSGRAD, and SPSA methods are implemented specifically in the Aqua module [30].

The Conjugate Gradient (CG) method [40] is an algorithm used for the numerical solution of systems of linear equations such as

$$Ax = b, \tag{3}$$

where x is the unknown vector, the known associated matrix A is symmetric, positive-definite and real, and b is also known. CG is often used to solve unconstrained optimization problems. Solving (3) is equivalent to minimizing a function

$$f(x) = \frac{1}{2}x^T Ax - x^T b, \tag{4}$$

since $f'(x) = Ax - b$. Therefore, if the objective function in (2) can be written as the one in (4), the CG method can be applied. For instance, this is the case of the Hamiltonian in (1). CG is an iterative algorithm that uses an initial guess to generate a sequence of improving approximate solutions for a problem, in which each approximation is derived from the previous ones. It only uses first derivatives. The CG method is also used in more sophisticated schemes such as the Truncated Newton (TNC) optimization method [41]. TNC computes, at every iteration k, the search direction d_k by applying the CG method to the Newton equations

$$\nabla^2_{xx} f(x_k) d_k = -\nabla f(x_k),$$

where f is the objective function. As stated above, the CG method is designed to solve systems with positive-definite matrices. In this case, the matrix A in (2) is the Hessian $\nabla^2_{xx} f$, and it may have negative eigenvalues when x_k is not close to a solution. Thus, the CG iteration is terminated as soon as a direction of negative curvature is generated. This adaptation of the CG method produces a search direction d_k that is a descent direction. Moreover, the adaptation guarantees that the fast convergence rate of the pure Newton method is preserved [42].

Limited-memory BFGS Bound (L-BFGS-B) [43] is an optimization algorithm of the family of the quasi-Newton methods. This means that, in contrast to Newton's method, it does not require the computation of the objective function's Hessian. L-BFGS-B uses the derivatives of f to identify the direction of steepest descent and to estimate the inverse Hessian matrix of f. While the original BFGS stores a dense $n \times n$ approximation to the inverse Hessian [44], L-BFGS-B stores only a few vectors that represent the approximation implicitly ("limited-memory"). Thus, the L-BFGS method is particularly well-suited for optimization problems which involve several variables. On the other hand, the BFGS method can be embedded in other methods such as the Sequential Least SQuares Programming (SLSQP) optimizer. This is a sequential quadratic programming algorithm (SQP) developed by Kraft in 1988 [45]. SLSQP can be used to minimize a function of several variables with any combination of equality and inequality constraints. For instance, we take $m = n = 1$ in (2). Then, the Lagrangian of the problem is[2]

$$\mathcal{L}(x, \lambda, \sigma) = f(x) - \lambda b(x) - \sigma c(x),$$

where λ and σ are Lagrange multipliers. At an iteration k, a sequential quadratic programming algorithm defines a search direction d_k as a solution to the quadratic programming subproblem

$$\begin{aligned} \min_{d} \quad & f(x_k) + \nabla f(x_k)^T d + \frac{1}{2} d^T \nabla^2_{xx} \mathcal{L}(x_k, \lambda_k, \sigma_k) d \\ \text{s.t.} \quad & b(x_k) + \nabla b(x_k)^T d \leq 0 \\ & c(x_k) + \nabla c(x_k)^T d = 0. \end{aligned}$$

[2]For the sake of simplicity, we write b and c instead of b_1 and c_1, respectively.

The Hessian of the Lagrangian $\nabla^2_{xx}\mathcal{L}(x_k, \lambda_k, \sigma_k)$ may not be easy to compute. In those cases, it is replaced by a quasi-Newton approximation such as a BFGS formula (see [46] for more details).

The Gaussian-Smoothed Line Search (GSLS) method is an implementation of the line search algorithm described in [47]. It uses gradient approximation based on Gaussian-smoothed samples on a sphere. The implementation of the GSLS method is specific to the Aqua module.

4.2. Gradient-free methods

For some optimization problems of the form (2), information about the derivative of the objective function f is unavailable, unreliable or impractical to obtain. For instance, f might be non-smooth, noisy or time-consuming to evaluate. So that, in those cases, methods that rely on derivatives may be of little use. We call gradient-free methods those optimization algorithms that do not use derivatives or finite differences.

The Constrained Optimization By Linear Approximation (COBYLA) is an optimization method for constrained problems where the derivatives of the objective function are not known [48, 49, 50]. It operates by evaluating the objective function and the constrains at the vertices of a trust region. Given an optimization problem of N variables, the trust region is a simplex of $N + 1$ vertices (a polytope of $N + 1$ vertices in N dimensions). Then, the values of the objective function are calculated at each vertex of the simplex. A linear polynomial is used to create an interpolation of the objective function. Similarly, a linear interpolation of the constraints is created. Another gradient-free method for which a simplex is constructed is the Nelder-Mead optimizer [51]. It is a direct search method (based on function comparison) which approximates a local optimum of a problem when the objective function varies smoothly and is unimodal. Nelder-Mead is initiated with a set of $N+1$ test points arranged as the vertices of a simplex. Then, it evaluates the objective function at each test point in order to find a new test point and to replace one of the old test points with the new one, and so the method progresses. Thus, the simplest approach for the evolution of the method is to replace the worst point by a point placed as the centroid of the remaining N points. If this new point is better than the best current point, then one can try stretching out exponentially the simplex along the line formed by the two points. On the other hand, if this new point is not much better than the previous value, it means that one is stepping across a valley, so the simplex has to be shrunk towards a better point. It has to be remarked that Nelder-Mead is a heuristic search method that, for some problems, may converge to non-stationary points [52].

The Powell algorithm is a conjugate direction method that finds local minima of a differentiable objective function f, even though no derivatives are taken [53]. Moreover, the function must be a real-valued function of a fixed number of real-valued inputs. A set of initial search vectors has to be defined. Typically, the N normals $\{s_1, \dots, s_N\}$ aligned to each axis are taken. Powell method minimizes f by a bi-directional search along each vector s_i. The line search along each vector can be done, for instance, by means of Brent's method [54]. Then, the minima found during each bi-directional line search are

$$\left\{ x_0 + \alpha_1 s_1, x_0 + \sum_{i=1}^{2} \alpha_i s_i, \dots, x_0 + \sum_{i=1}^{N} \alpha_i s_i \right\},$$

where x_0 is the point and α_i is the scalar obtained by the bi-directional search along s_i. A new position is expressed as a linear combination of the search vectors $x_1 = x_0 + \sum_{i=1}^{N} \alpha_i s_i$, and the new displacement vector $\sum_{i=1}^{N} \alpha_i s_i$ becomes a new search vector added to the end of the search vector list. Meanwhile, the search vector which contributed most to the new direction, i.e., $i_d = \arg\max_{i=1}^{N} |\alpha_i| \|s_i\|$, is deleted from the search vector list. Thus, the new set of N search vectors is

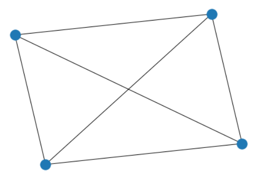
$$\left\{ s_1, \dots, s_{i_{d-i}}, s_{i_{d+i}}, \dots s_N, \sum_{i=1}^{N} \alpha_i s_i \right\}.$$

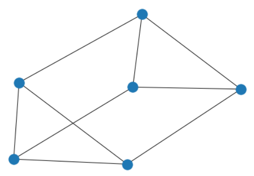
The algorithm iterates until it reaches a stopping test.

The Nakanishi-Fujii-Todo (NFT) optimization method is specialized for quantum-classical hybrid algorithms based on parameterized quantum circuits. The optimization problem has to satisfy three conditions specified in Section II of [55] about the quantum circuit employed in the implementation. On the other hand, the implementation of the NFT method that we use is specific to the Aqua module [30].

5. Numerical experiments

To test the performance of the different classical optimizers that we are considering in this work, we have conducted a series of experiments on quantum simulators. We have focused on the Max-Cut problem since, in


(a) 3-regular graph of 4 vertices (K_4)


(b) 3-regular graph of 6 vertices (G6a)

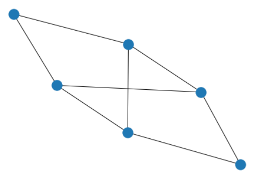

(c) 3-regular graph of 6 vertices (G6b)

Figure 1: 3-regular graphs of up to 6 vertices

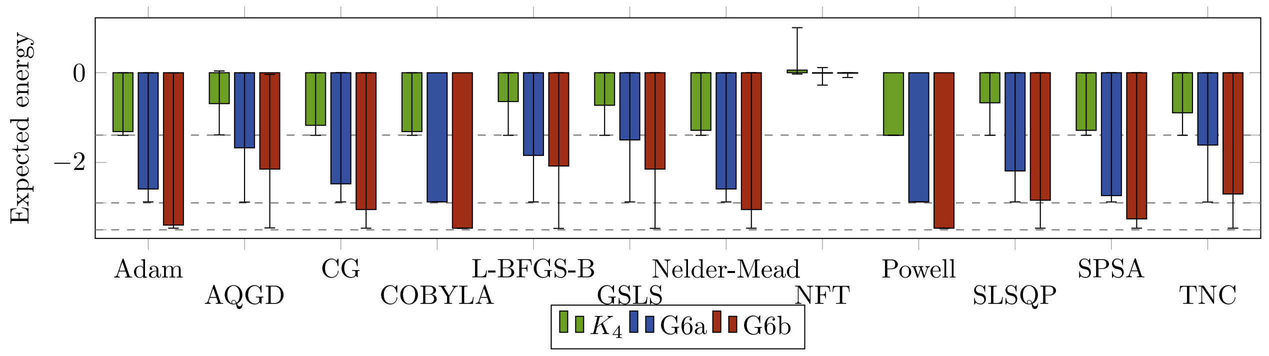


Figure 2: Expected energy with $p = 1$. The theoretical target values are shown in gray dashed lines, and the error bars correspond to the minimum and maximum values reached

addition to being NP-hard, it has been widely studied in connection to QAOA. We have started by studying 3-regular graphs (that is, graphs whose vertices all have degree exactly 3), which were the particular graph family for which the properties of QAOA were first investigated [10] and have been used widely since then [12]. There are exactly 3 graphs of up to 6 vertices which are 3-regular: one of size 4 and two of size 6 (see Figure 1). We have run QAOA 50 times using the default parameters of IBM's Qiskit Aqua implementation with the statevector simulator and the 12 classical optimizers described in Section 4 on the graphs of sizes 4 and 6, with values of $p = 1, 2, 3$. We have computed the average of both the expected energy of the final state obtained by QAOA and of the computation time needed to obtain it. The errors of the expected energies are also reported. The results for $p = 1$ are shown in Figure 2, for $p = 2$ in Figure 3, and for $p = 3$ in Figure 4. The computational times for $p = 1$ can be found in Figure 5, whereas the times of both $p = 2, 3$ are combined together in Figure 6.

As it can be appreciated in Figures 2, 3 and 4, Adam, CG, COBYLA, Nelder-Mead, Powell and SPSA consistently obtain the lowest energy values, while NFT method produces the worst results among all the considered methodologies. On the other hand, the results of AQGD, L-BFGS-B, GSLS, SLSQP and TNC (which are all gradient-based methods) suggest either a tuning problem for these algorithms or an intrinsic difficulty of the graphs studied: the maximum and minimum values these optimizers attain correspond to zero and the theoretically predicted result, respectively. In fact, we have observed that L-BFGS-B, GSLS, SLSQP and TNC oscillate between one and the other value, with no intermediate results, regardless of the choice of parameters. Besides, AQGD takes solutions in the whole range of values and increasing either the time step or the momentum the performance is improved and the optimizer does not get trapped in zero. Of course, this leads to an increase of the computational cost. These results could be explained by the particular form of the analytical expressions of the expected energy. In fact, it has been noted [11] that these energies usually are hard to optimize because of their non-convexity and their many local minima. To illustrate this, we have applied Theorem 1 to the graphs that we are using in our experiments to obtain the formulas in Table 1. More in particular, in Figure 7 we show a contour of the K_4 graph plot in which one can observe several minima and maxima, and saddle points of value 0 between them which might lead some optimizers to get trapped there. Anyhow, it has to be remarked also that this behaviour is much more notorious for $p = 1, 2$ than for $p = 3$. Moreover, in the case of $p = 1$, Adam, CG, COBYLA, Nelder-Mead (which is gradient-free) and SPSA, even though they produce good average results, also show the oscillatory behaviour described above.

Gradient methods such as Adam or SPSA show computational times which are up to one or even two orders

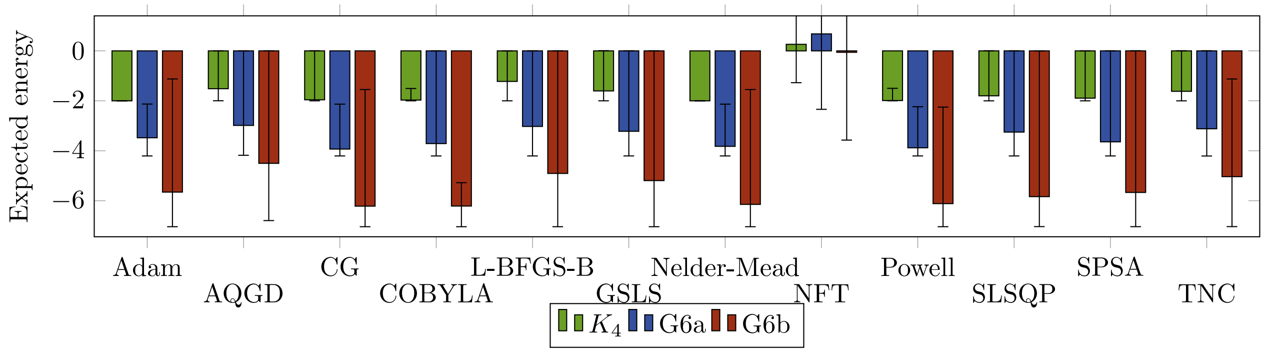


Figure 3: Expected energy with $p = 2$. The error bars correspond to the minimum and maximum values reached

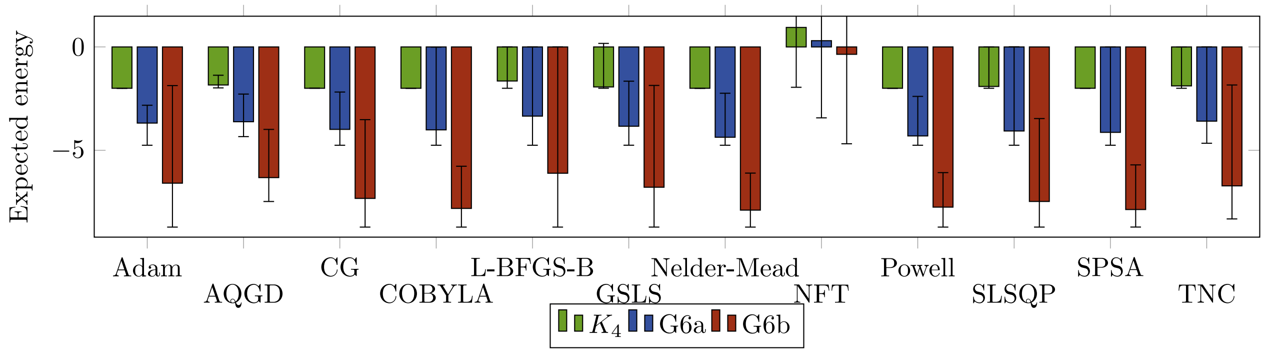


Figure 4: Expected energy with $p = 3$. The error bars correspond to the minimum and maximum values reached

of magnitude higher than those of the gradient-free methods such as COBYLA, Nelder-Mead or Powell (see Figures 5 and 6), which obtain similarly good energy values. For this reason, we have repeated our experiments with the six 3-regular graphs of size 8 (see Figure 8) only with five of the optimizers: CG, COBYLA, Nelder-Mead, Powell and SLSQP. These optimizers were selected because of the quality of the solutions they attain in a reasonable time or, as for instance in the case of SLSQP, because of their speed. With them, we have run again the QAOA algorithm 50 times with $p = 1, 2, 3$. The average energies are shown in Figure 9 and the computational times in Figure 10. The lowest energy energy values are obtained by COBYLA and Powell, with Nelder-Mead being a serious contender especially for $p = 2, 3$. These methods, however, have an execution time that is three or four times slower than CG or SLSQP (which is, again, the fastest of them all).

Another interesting thing to notice is that the analytical expression for the expected energy of graphs $G8a$ and $G8b$ is exactly the same (see Table 1). This follows from the expression of Theorem 1, the fact that neither $G8a$ nor $G8b$ have any triangle and the discussion just before Lemma 1. However, the circuit that is used by QAOA is different for both graphs (because it dependes directly on the cost Hamiltonian) and it seems to be more difficult for the optimizers in the case of $G8b$, since they take about 5 or 6 times longer to find a solution for it (see Figure 10).

To validate the conclusions we have obtained so far, we have run a new series of experiments with 5 graphs of 6 vertices with no restriction on their degree, i.e., graphs that are not necessarily 3-regular. Among the 154 non-isomorphic graphs of size 6, we have selected 5 of them at random (see Figure 11) and run 50 instances of QAOA with the 12 classical optimizers, again with values of p equal to 1, 2 and 3. In Figure 12, the expected energies are averaged for the 5 randomly selected graphs. As not all graphs attain the same value for their maximal cut, in order to do a reasonable comparison, we define the mean value

$$\mu_{\text{opt}} = \left\langle \frac{\mu_{\text{opt,graph}}}{|\min_{\text{graph}}|} \right\rangle_{\text{opt}}, \tag{5}$$

where the mean obtained by each optimizer method in each graph $\mu_{\text{opt,graph}}$ is normalized by the minimum

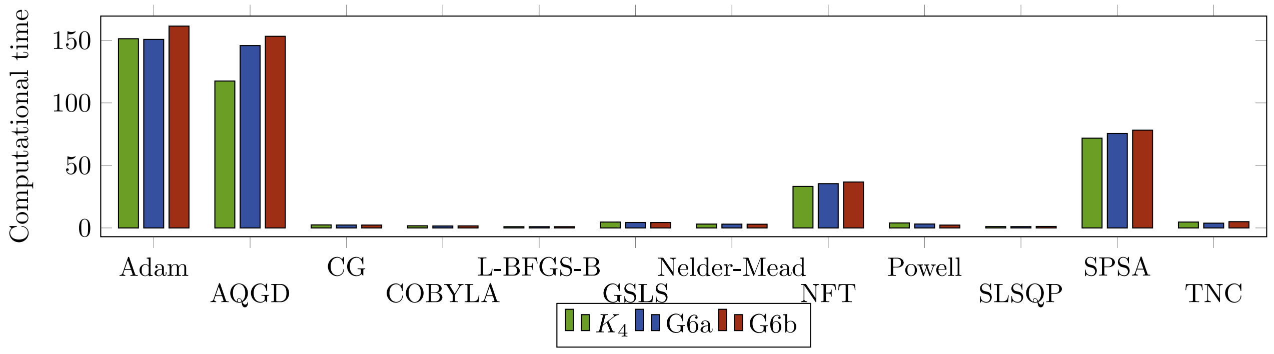


Figure 5: Computational times for the graphs with $p = 1$. The values are normalized by dividing the computational times by the time obtained by the fastest optimization method, which in this case is L-BFGS-B for the three considered graphs

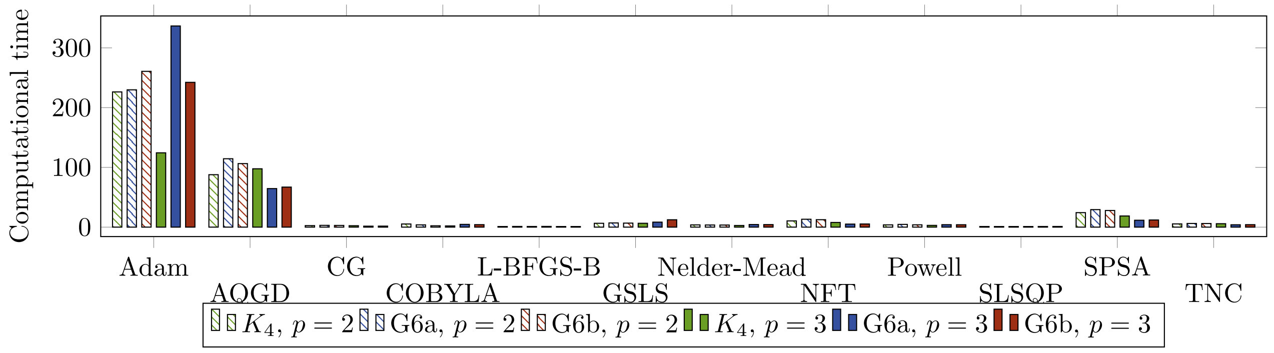


Figure 6: Computational times for the graphs with $p = 2, 3$. The values are normalized by dividing the computational times by the time obtained by the fastest optimization method: (i) if $p = 2$, L-BFGS-B for K_4, and SLSQP for G6a and G6b; and (ii) if $p = 3$, SLSQP for K_4, and L-BFGS-B for G6a and G6b

value $\min_{\text{graph}}$ of that graph (over all possible cuts). On the other hand, in Figure 12 the average times of each method are presented. No normalization is introduced in this case, since we are interested in comparing the average time that would employ the different methods to solve any given problem. These latest results confirm the trends observed in the 3-regular case. The relatively small standard deviations, specially for the expected energies in Figure 12, show that, after normalization, there is no remarkably different behaviour among the five chosen graphs.

All the above experiments were carried out with Qiskit statevector simulator, so they assume a perfect, error-free behaviour of the quantum computer, where noise only appears in the evaluation of the function to be optimized as a consequence of the stochastical nature of quantum measurement. However, in practice, quantum computers are noisy and subject to errors, so we have conducted some experiments to test whether the performance we have observed also carries out to a situation in which noise is present. These experiments were done with Qiskit's qasm simulator using the noise model of IBM Quantum Experience [56] real quantum computers: IBMQ Rome. This computer only has 5 qubits, so the only graph that we can use is K_4, the 3-regular graph with 4 vertices. As in the experiments with the statevector simulator, we have run 50 instances of each optimizer with $p = 1, 2, 3$. However, the results with $p = 2, 3$ do not improve over the results obtained with $p = 1$ and take much longer, so we report only these latter data in Figure 13. The reason for the experiments with higher p to perform worse than the ones with $p = 1$ is that the size of the circuits is bigger, so the gate errors and the noise have a stronger effect on the quality of the results (this behaviour is consistent with what other authors have previously observed [16]). Notice that the results confirm our previous observations for the noiseless case, with Adam, CG, COBYLA, Nelder-Mead, Powell and SPSA reaching the lowest values of energy, but with Adam and SPSA being orders of magnitude slower than the other methods.

Graph	**Expected energy as a function of QAOA's angles β and γ**
K_4	$12\,\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^2+12\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
$G6a$	$6\,\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^2+18\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
$G6b$	$6\,\cos(2\beta)\cos(2\gamma)^3\sin(2\beta)\sin(2\gamma)-2\,\cos(2\beta)\cos(2\gamma)\sin(2\beta)\sin(2\gamma)^3$ $+14\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)-\cos(2\beta)\sin(2\beta)\sin(2\gamma)^3$ $+2\,\cos(2\beta)\cos(2\gamma)\sin(2\beta)\sin(2\gamma)$
$G8a$	$24\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
$G8b$	$24\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
$G8c$	$6\,\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^2+24\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
$G8d$	$12\,\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^2+24\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
$G8e$	$3\,\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^2+24\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
$G8f$	$24\,\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^2+24\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
R_1	$8\,\cos(2\beta)\cos(2\gamma)^3\sin(2\beta)\sin(2\gamma)+8\,\cos(2\beta)\cos(2\gamma)\sin(2\beta)\sin(2\gamma)$
R_2	$8\,\cos(2\gamma)^3\sin(2\beta)^2\sin(2\gamma)^2+8\,\cos(2\beta)\cos(2\gamma)^3\sin(2\beta)\sin(2\gamma)$ $+4\,\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^2+12\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$
R_3	$4\,\cos(2\gamma)^6\sin(2\beta)^2\sin(2\gamma)^2+4\,\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^6$ $+18\,\cos(2\gamma)^5\sin(2\beta)^2\sin(2\gamma)^2+6\,\cos(2\gamma)\sin(2\beta)^2\sin(2\gamma)^6+9\,\cos(2\gamma)^4\sin(2\beta)^2\sin(2\gamma)^2$ $+3\,\sin(2\beta)^2\sin(2\gamma)^6+10\,\cos(2\beta)\cos(2\gamma)^4\sin(2\beta)\sin(2\gamma)+2\,\cos(2\gamma)^3\sin(2\beta)^2\sin(2\gamma)^2$ $+12\,\cos(2\beta)\cos(2\gamma)^3\sin(2\beta)\sin(2\gamma)+2\,\cos(2\beta)\cos(2\gamma)\sin(2\beta)\sin(2\gamma)$
R_4	$\cos(2\gamma)^3\sin(2\beta)^2\sin(2\gamma)^2+4\,\cos(2\beta)\cos(2\gamma)^3\sin(2\beta)\sin(2\gamma)$ $+\cos(2\gamma)^2\sin(2\beta)^2\sin(2\gamma)^2+6\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)$ $+\cos(2\gamma)\sin(2\beta)^2\sin(2\gamma)^2+6\,\cos(2\beta)\cos(2\gamma)\sin(2\beta)\sin(2\gamma)$
R_5	$3\,\cos(2\beta)\cos(2\gamma)^2\sin(2\beta)\sin(2\gamma)+8\,\cos(2\beta)\cos(2\gamma)\sin(2\beta)\sin(2\gamma)$ $+\cos(2\beta)\sin(2\beta)\sin(2\gamma)$

Table 1: Analytical expressions of the expected energy of the QAOA states when $p = 1$ for the different graphs considered in our experiments

6. Conclusions and future work

In this paper, we have conducted a study of the influence of the choice of classical optimizer in the performance of the Quantum Approximate Optimization Algorithm. We have run experiments with the Max-Cut problem and different sizes of graphs (both 3-regular and not) using a quantum simulator with and without noise. We have used the 12 optimizers included in the Aqua module of IBM's Qiskit framework, which is more than twice the number of minimizers studied in previous works.

Our results show that the optimizers that obtain the solutions with the lowest energy value are, consistently across all experiments, Adam, CG, COBYLA, Nelder-Mead, Powell and SPSA. However, of them, two of the gradient-based methods, Adam and SPSA, are orders of magnitude slower than the others. Since QAOA is a quantum algorithm and, hence, its results are probabilistic, it makes sense to use methods which are fast while obtaining good quality solutions. From this point of view, the more balanced methods have proven to be COBYLA and Powell, but even SLSQP may be a competitive option. Indeed, notice that since QAOA is a probabilistic method, it is usually run several times and the best solution among all the executions is returned. Then, since SLQSP is the fastest of all the methods under study, it can perform several more executions in the same time, giving it more chances of obtaining lower energy values.

Our experiments also showed some interesting behaviours of the optimizers. On the one hand, several of them present results that oscillate between the optimal value and energies close to 0. This suggests that they are being trapped in local minima. We would like to study these anomalies in more detail and, if possible, to determine for which types of graphs this is more likely to happen.

On the other hand, we have shown that even for graphs whose analytical energy formula is exactly the same, the behaviour of the optimizers can be very different, especially in total running time. This is caused by the direct use in the Quantum Approximate Optimization Algorithm of the Hamiltonian that encodes the problem.

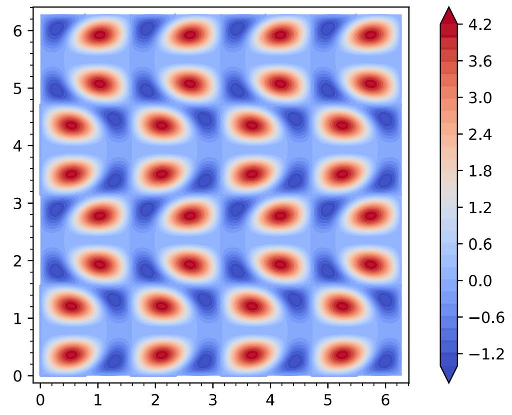
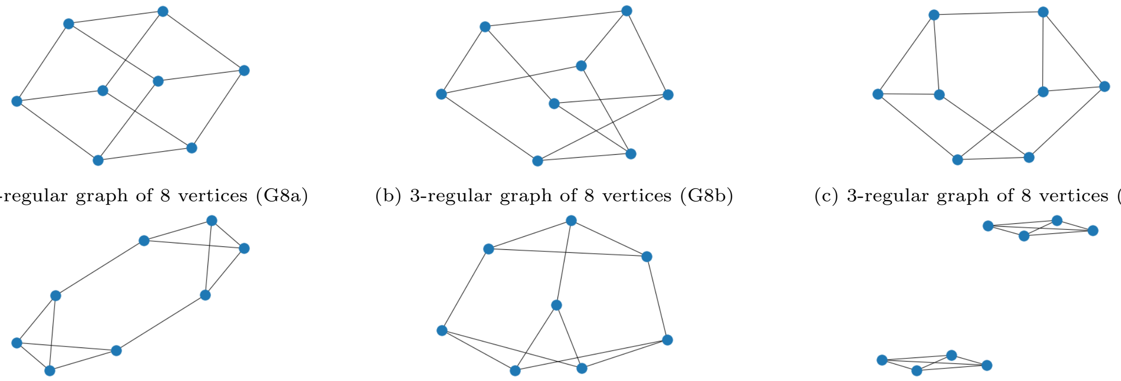


Figure 7: Contour plot of expected energy for the K_4 QAOA states when $p = 1$



(a) 3-regular graph of 8 vertices (G8a) (b) 3-regular graph of 8 vertices (G8b) (c) 3-regular graph of 8 vertices (G8c)

(d) 3-regular graph of 8 vertices (G8d) (e) 3-regular graph of 8 vertices (G8e) (f) 3-regular graph of 8 vertices (G8f)

Figure 8: 3-regular graphs of 8 vertices

This opens, then, two interesting avenues for future investigation. First, the possibility of using information (even if partial) from the analytical energy function to direct the search of the optimizers. Second, the use of quantum circuit simplifications and equivalences to reduce the execution time when possible.

Acknowledgments

This work was supported in part by the MINECO under Grant MTM-2017-83506-C2-2-P, and in part by the MICINN under Grant RTI2018-098085-B-C44, Grant FC-GRUPIN-IDI/2018/000193, and under Grant FC-GRUPIN-IDI/2018/000226.

References

[1] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information: 10th Anniversary Edition, Cambridge University Press, 2011.

[2] P. Shor, Algorithms for quantum computation: Discrete logarithms and factoring, Proceedings of FOCS (1994) 124–134.

[3] L. K. Grover, A fast quantum mechanical algorithm for database search, in: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, STOC '96, ACM, NY, USA, 1996, pp. 212–219.

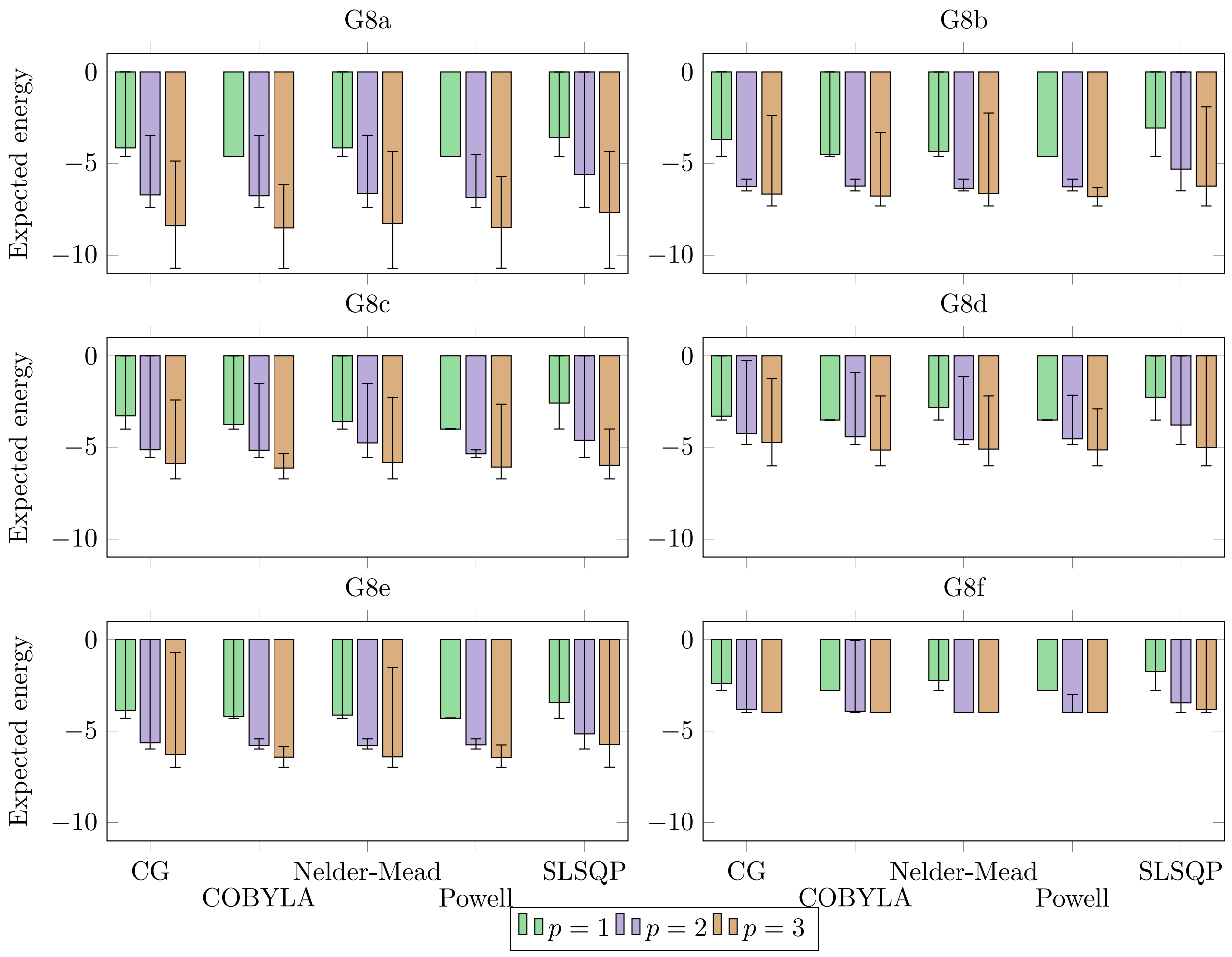


Figure 9: Expected energies of the 6 3-regular graphs of size 8 for $p = 1, 2, 3$. The error bars correspond to the minimum and maximum values reached

[4] S. E. Venegas-Andraca, Quantum Walks for Computer Scientists, Synthesis Lectures on Quantum Computing, Morgan & Claypool Publishers, 2008.

[5] E. F. Combarro, J. Ranilla, I. Rúa, Quantum walks for the determination of commutativity of finite dimensional algebras, Journal of Computational and Applied Mathematics 354 (2019) 496–506.

[6] E. Farhi, J. Goldstone, S. Gutmann, M. Sipser, Quantum computation by adiabatic evolution, arXiv:quant-ph/0001106v1, 2000.

[7] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, D. Preda, A quantum adiabatic evolution algorithm applied to random instances of an NP-Complete problem, Science 292 (5516) (2001) 472–475.

[8] C. C. McGeoch, Adiabatic Quantum Computation and Quantum Annealing, Synthesis Lectures on Quantum Computing, Morgan&Claypool Publishers, 2014.

[9] D-Wave Quantum Computers, https://www.dwavesys.com/.

[10] E. Farhi, J. Goldstone, S. Gutmann, A quantum approximate optimization algorithm, arXiv:1411.4028 (2014).

[11] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, M. D. Lukin, Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices, Phys. Rev. X 10 (2020) 021067.

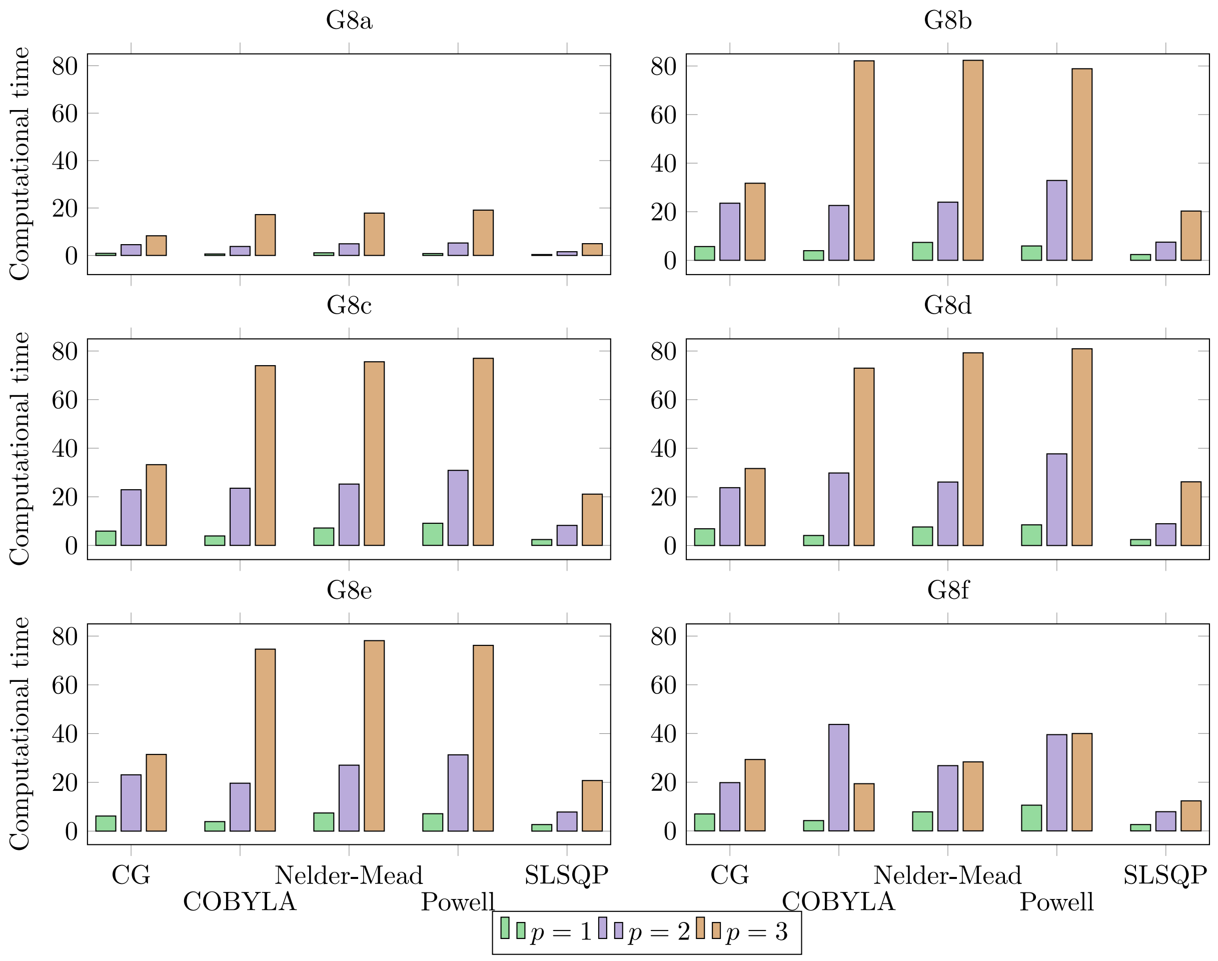


Figure 10: Computational times, in seconds, on the six 3-regular graphs of size 8 for $p = 1, 2, 3$

[12] G. G. Guerreschi, A. Y. Matsuura, QAOA for max-cut requires hundreds of qubits for quantum speed-up, Scientific reports 9 (1) (2019) 6903.

[13] R. Shaydulin, Y. Alexeev, Evaluating quantum approximate optimization algorithm: A case study, in: 2019 Tenth International Green and Sustainable Computing Conference (IGSC), IEEE, 2019, pp. 1–6.

[14] G. E. Crooks, Performance of the quantum approximate optimization algorithm on the maximum cut problem, arXiv preprint arXiv:1811.08419 (2018).

[15] M. Willsch, D. Willsch, F. Jin, H. D. Raedt, K. Michielsen, Benchmarking the quantum approximate optimization algorithm, Quantum Inf. Process. 19 (7) (2020) 197.

[16] M. Alam, A. Ash-Saki, S. Ghosh, Analysis of quantum approximate optimization algorithm under realistic noise in superconducting qubits, arXiv preprint arXiv:1907.09631 (2019).

[17] M. Streif, M. Leib, Forbidden subspaces for level-1 QAOA and IQP circuits, arXiv:2007.13563 (2020).

[18] S. Hadfield, Z. Wang, E. G. Rieffel, B. O'Gorman, D. Venturelli, R. Biswas, Quantum approximate optimization with hard and soft constraints, in: Proceedings of the Second International Workshop on Post Moores Era Supercomputing, PMES'17, Association for Computing Machinery, NY, USA, 2017, p. 15–21.

[19] J. Ostroswki, R. Herrman, T. S. Humble, G. Siopsis, Lower bounds on circuit depth of the quantum approximate optimization algorithm, arXiv preprint arXiv:2008.01820 (2020).

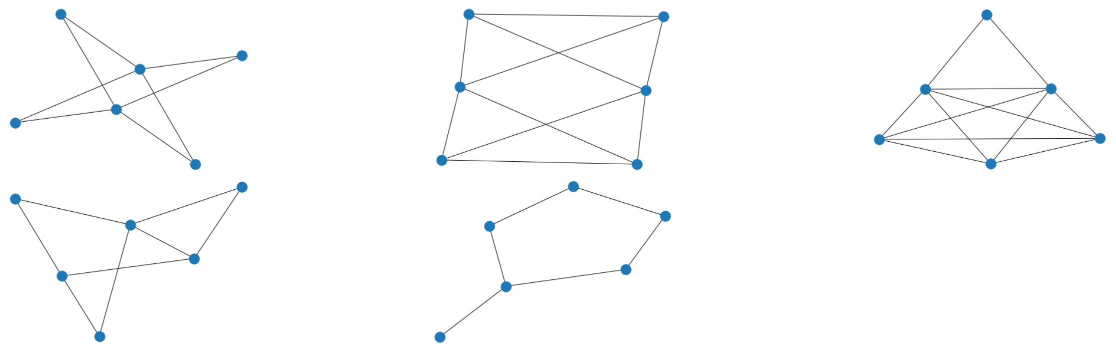


Figure 11: Five randomly chosen graphs of size 6 and degree bounded by 3

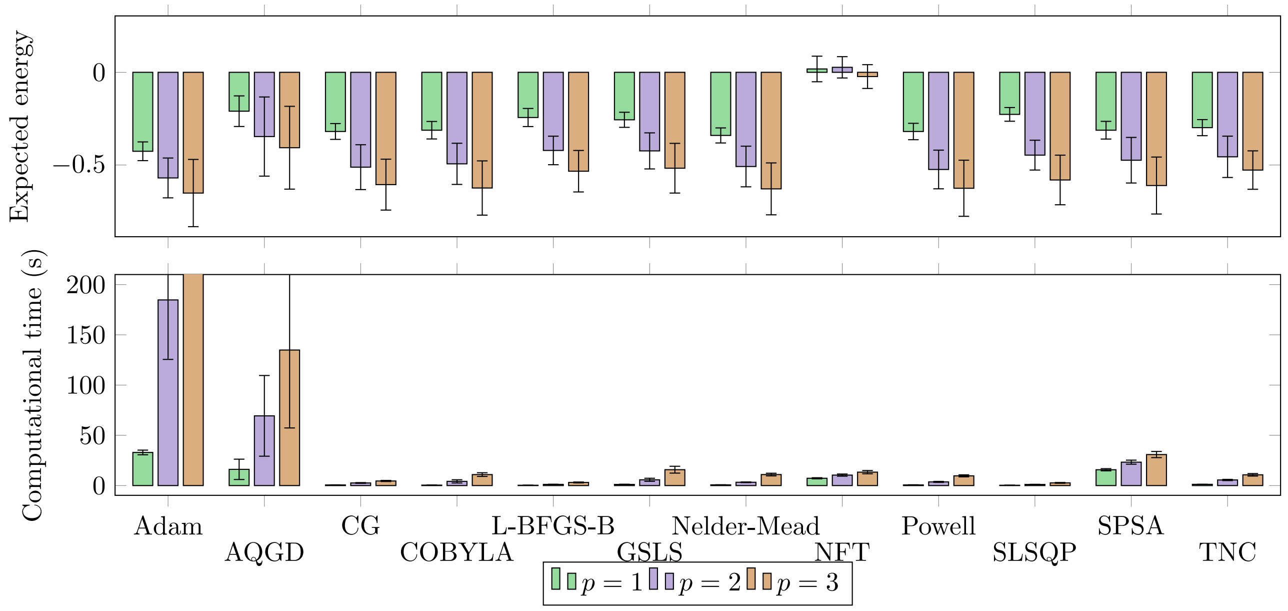


Figure 12: Results for the five random graphs in Figure 11. (Upper panel) Expected energy averaged as explained in (5). The error bars correspond to the standard deviation associated to such average, which is normalized in the same manner. (Lower panel) Computational times averaged over the five graphs. The error bars correspond to the standard deviation associated to such average

[20] P. Vikstål, M. Grönkvist, M. Svensson, M. Andersson, G. Johansson, G. Ferrini, Applying the quantum approximate optimization algorithm to the tail assignment problem, arXiv:1912.10499 (2019).

[21] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, S. Boixo, M. Broughton, B. B. Buckley, D. A. Buell, et al., Quantum approximate optimization of non-planar graph problems on a planar superconducting processor, arXiv preprint arXiv:2004.04197 (2020).

[22] G. G. Guerreschi, M. Smelyanskiy, Practical optimization for hybrid quantum-classical algorithms, arXiv preprint arXiv:1701.01450 (2017).

[23] D. Wierichs, C. Gogolin, M. Kastoryano, Avoiding local minima in variational quantum eigensolvers with the natural gradient optimizer, arXiv preprint arXiv:2004.14666 (2020).

[24] G. Nannicini, Performance of hybrid quantum-classical variational heuristics for combinatorial optimization, Physical Review E 99 (1) (2019) 013304.

[25] M. Suzuki, Generalized trotter's formula and systematic approximants of exponential operators and inner

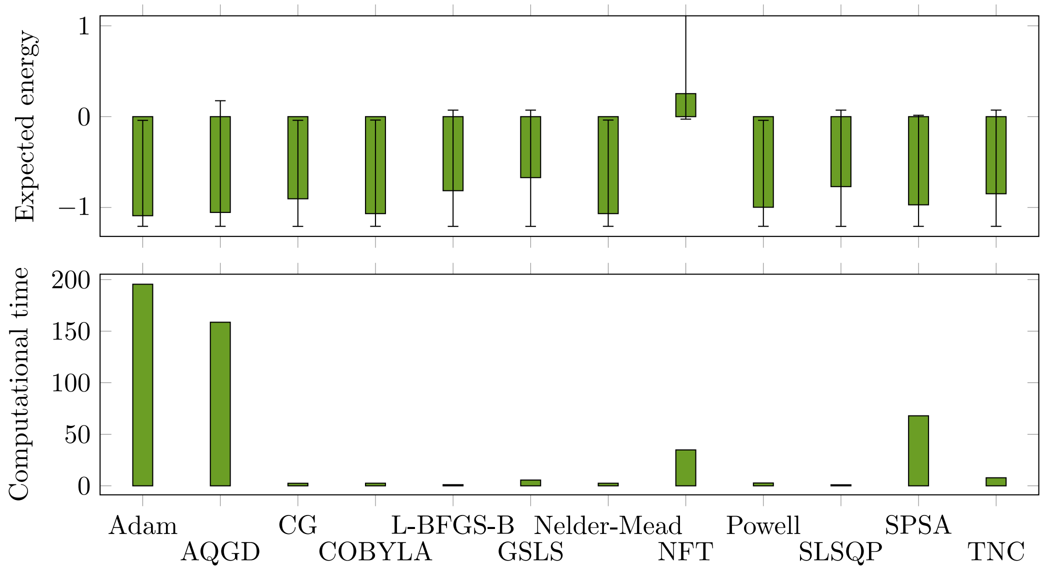


Figure 13: Results for K_4 with $p = 1$ in the noisy quantum simulation. (Upper panel) Expected energy. The error bars correspond to the minimum and maximum values reached. (Lower panel) Computational times. The values are normalized by dividing the computational times by the time obtained by the fastest optimization method, which in this case is SLSQP

derivations with applications to many-body problems, Communications in Mathematical Physics 51 (2) (1976) 183–190.

[26] M. Deza, M. Laurent, Applications of cut polyhedra—ii, Journal of Computational and Applied Mathematics 55 (2) (1994) 217–247.

[27] S. Poljak, Z. Tuza, Maximum cuts and large bipartite subgraphs, DIMACS Series 20 (1995) 181–244.

[28] R. Karp, Reducibility among combinatorial problems, in: R. Miller, J. Thatcher (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.

[29] M. Yannakakis, Node-and edge-deletion np-complete problems, in: Proceedings of the tenth annual ACM symposium on Theory of computing, 1978, pp. 253–264.

[30] Qiskit Aqua 0.7.3, https://github.com/Qiskit/qiskit-aqua/releases/tag/0.7.3.

[31] H. Abraham, AduOffei, R. Agarwal, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, et al., Qiskit: An open-source framework for quantum computing (2019). doi:10.5281/zenodo.2562110.

[32] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, et al., SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272.

[33] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980 (2014).

[34] L. Bottou, O. Bousquet, The tradeoffs of large scale learning, in: Advances in neural information processing systems, 2008, pp. 161–168.

[35] S. J. Reddi, S. Kale, S. Kumar, On the convergence of adam and beyond, arXiv:1904.09237 (2019).

[36] K. Mitarai, M. Negoro, M. Kitagawa, K. Fujii, Quantum circuit learning, Physical Review A 98 (3) (2018) 032309.

[37] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, N. Killoran, Evaluating analytic gradients on quantum hardware, Physical Review A 99 (3) (2019) 032331.

[38] J. C. Spall, Multivariate stochastic approximation using a simultaneous perturbation gradient approximation, IEEE transactions on automatic control 37 (3) (1992) 332–341.

[39] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, J. M. Gambetta, Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets, Nature 549 (7671) (2017) 242–246.

[40] M. R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of research of the National Bureau of Standards 49 (6) (1952) 409–436.

[41] S. G. Nash, A survey of truncated-newton methods, Journal of Computational and Applied Mathematics 124 (1) (2000) 45 – 59, numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[42] J. Nocedal, S. Wright, Numerical optimization, Springer Science & Business Media, 2006.

[43] R. H. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization, SIAM Journal on Scientific Computing 16 (5) (1995) 1190–1208.

[44] R. Fletcher, Practical methods of optimization, John Wiley & Sons, 2013.

[45] D. Kraft, A software package for sequential quadratic programming, Technical Report DFVLR-FB 88-28 (1988).

[46] J. Nocedal, S. J. Wright, Sequential Quadratic Programming, Springer, New York, 2006, pp. 529–562.

[47] A. S. Berahas, L. Cao, K. Choromanski, K. Scheinberg, A theoretical and empirical comparison of gradient approximations in derivative-free optimization, arXiv:1905.01332 (2019).

[48] M. J. D. Powell, A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation, Springer Netherlands, Dordrecht, 1994, pp. 51–67.

[49] M. J. D. Powell, Direct search algorithms for optimization calculations, Acta Numerica 7 (1998) 287–336.

[50] M. J. D. Powell, A view of algorithms for optimization without derivatives, Mathematics Today-Bulletin of the Institute of Mathematics and its Applications 43 (5) (2007) 170–174.

[51] J. A. Nelder, R. Mead, A Simplex Method for Function Minimization, The Computer Journal 7 (4) (1965) 308–313.

[52] K. I. M. McKinnon, Convergence of the Nelder–Mead simplex method to a nonstationary point, SIAM Journal on Optimization 9 (1) (1998) 148–158.

[53] M. J. D. Powell, An efficient method for finding the minimum of a function of several variables without calculating derivatives, The Computer Journal 7 (2) (1964) 155–162.

[54] R. P. Brent, Algorithms for minimization without derivatives, Courier Corporation, 2013.

[55] K. M. Nakanishi, K. Fujii, S. Todo, Sequential minimal optimization for quantum-classical hybrid algorithms, arXiv:1903.12166 (2019).

[56] IBM Q Experience, https://www.research.ibm.com/ibm-q/.