Application of the Quantum Approximate Optimization Algorithm to the MaxCut problem

Quantum Information Project, AP3421-PR

Santiago Vallés-Sanclemente, Isacco Gobbi, Olexiy Fedorets

13th February 2020

Table of Contents

Why QAOA?

- Proposed by E. Farhi and J. Goldstone in 2014
- Approximates solutions to NP-hard optimization problems (for some time better than classical algorithms)

Why QAOA?

- Proposed by E. Farhi and J. Goldstone in 2014
- Approximates solutions to NP-hard optimization problems (for some time better than classical algorithms)
- Algorithm suitable for noisy intermediate-scale quantum computers (NISQ) $\rightarrow$ low circuit-depth, small number of qubits
- Only one qubit per variable required: hard to beat!

What is QAOA? - Quantum Approximate Optimization Algorithm

- Combinatorial optimization problems: finding an optimal object from a finite set
- An approximation algorithm returns a solution to a combinatorial optimization problem that is provably close to optimal

What is QAOA? - Quantum Approximate Optimization Algorithm

- Combinatorial optimization problems: finding an optimal object from a finite set
- An approximation algorithm returns a solution to a combinatorial optimization problem that is provably close to optimal
- Optimality defined with respect to some target function C(z) of an n-bit string $z \in \{0,1\}^n$ that needs to be maximized
- m-clause target function $C(z) = \sum_{k=1}^{m} C_k(z)$

  $C_k(z) = +1$ if z satisfies clause k

  $C_k(z) = 0$ if z does not satisfy clause k

How does QAOA work?[1]

- Circuit consists of alternating cost unitary $\hat{U}_C(\gamma) = \exp(-i\gamma C(z))$ and mixing unitary $\hat{U}_B(\beta) = \exp\left(-i\beta \sum_{j=1}^{n} \hat{X}_j\right)$

[1] Farhi et al., "A Quantum Approximate Optimization Algorithm", 2014

How does QAOA work?[1]

- Circuit consists of alternating cost unitary $\hat{U}_C(\gamma) = \exp(-i\gamma C(z))$ and mixing unitary $\hat{U}_B(\beta) = \exp\left(-i\beta \sum_{j=1}^{n} \hat{X}_j\right)$
- Build variational state: $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle = \hat{U}_B(\beta_p)\hat{U}_C(\gamma_p)\ldots\hat{U}_B(\beta_1)\hat{U}_C(\gamma_1)\,|+\rangle^{\otimes n}$

[1] Farhi et al., "A Quantum Approximate Optimization Algorithm", 2014

How does QAOA work?[1]

- Circuit consists of alternating cost unitary $\hat{U}_C(\gamma) = \exp(-i\gamma C(z))$ and mixing unitary $\hat{U}_B(\beta) = \exp\left(-i\beta \sum_{j=1}^{n} \hat{X}_j\right)$
- Build variational state:
  $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle = \hat{U}_B(\beta_p)\hat{U}_C(\gamma_p) \ldots \hat{U}_B(\beta_1)\hat{U}_C(\gamma_1) \, |+\rangle^{\otimes n}$
- Measure expectation value $\langle\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})|C(z)|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle$

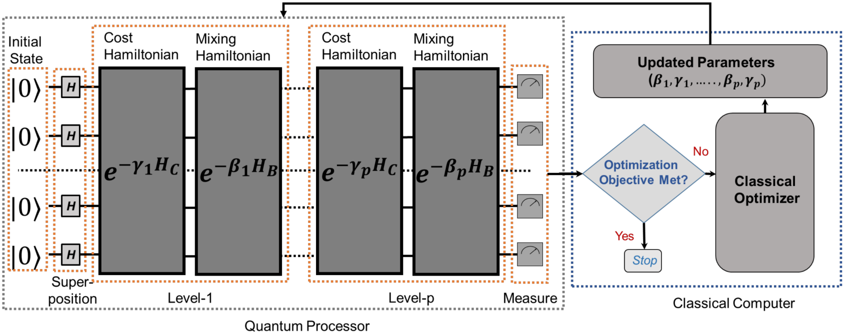
[1] Farhi et al., "A Quantum Approximate Optimization Algorithm", 2014

How does QAOA work?[1]

- Circuit consists of alternating cost unitary $\hat{U}_C(\gamma) = \exp(-i\gamma C(z))$ and mixing unitary $\hat{U}_B(\beta) = \exp\left(-i\beta \sum_{j=1}^{n} \hat{X}_j\right)$
- Build variational state: $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle = \hat{U}_B(\beta_p)\hat{U}_C(\gamma_p) \ldots \hat{U}_B(\beta_1)\hat{U}_C(\gamma_1)\ |+\rangle^{\otimes n}$
- Measure expectation value $\langle\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})|C(z)|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle$
- Explore the solution space of $2p$ angles $\boldsymbol{\beta} = (\beta_1, .., \beta_p) \in [0, \pi]^p,\ \boldsymbol{\gamma} = (\gamma_1, .., \gamma_p) \in [0, 2\pi]^p$ to maximize $\langle C(z)\rangle$
- Guaranteed to find global maximum of $\langle C(z)\rangle$ for $p \to \infty$

[1] Farhi et al., "A Quantum Approximate Optimization Algorithm", 2014

The QAOA circuit[2]



[2] Alam et al., "Analysis of Quantum Approximate Optimization Algorithm under Realistic Noise in Superconducting Qubits", 2019

Classical optimization - Differential Evolution[3]

- DE-algorithm performs global optimization of a function via genetic evolution

[3] Sribonchandr, "Improved Differential Evolution Algorithm for Flexible Job Shop Scheduling Problems", 2019

[3] Storn and Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces", 2019

Classical optimization - Differential Evolution[3]

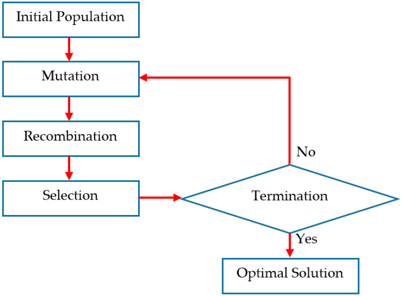
- DE-algorithm performs global optimization of a function via genetic evolution
- Evaluates each solution candidate until the variation of the function is under the given threshold

[3] Sriboonchandr, "Improved Differential Evolution Algorithm for Flexible Job Shop Scheduling Problems", 2019

[3] Storn and Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces", 2019

Classical optimization - Differential Evolution[3]

- DE-algorithm performs global optimization of a function via genetic evolution
- Evaluates each solution candidate until the variation of the function is under the given threshold
- Parallelizable!
- Numerous algorithm parameters like mutation strength, population size etc. need to be tuned tediously



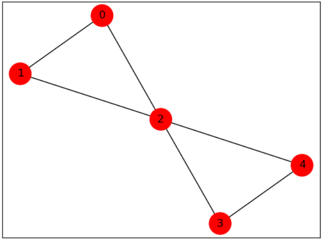
[3] Sriboonchandr, "Improved Differential Evolution Algorithm for Flexible Job Shop Scheduling Problems", 2019

[3] Storn and Price, "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces", 2019

Table of Contents

MaxCut

- Graph: $G = (V, E)$ where V is the set of nodes of the graph and E is the set of edges
- Cut: partition of the vertices set V into two disjoint subsets S and $\overline{S}$
- `MaxCut`: find a cut that crosses the greatest number of edges



V={0,1,2,3,4}
E={(2,0),(2,1),(2,3),
(2,4),(0,1),(3,4)}

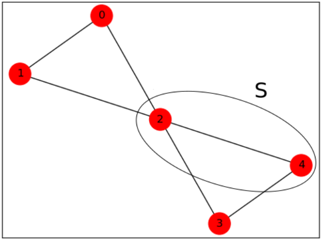
MaxCut: cost function

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

Any cut divides V in two subsets S and $\overline{S}$ such that $S \cap \overline{S} = \emptyset$ and $S \cup \overline{S} = V$

- Input: $z = \{z_1, z_2, ..., z_n\}$ string of S:
  $z_i = +1$ if node $i \in S$
  $z_i = -1$ if node $i \in S$
- Output: number of crossed edges

MaxCut: an example



Cut $S = \{2, 4\}$
$\Rightarrow z = \{-1, -1, +1, -1, +1\}$

$$
\begin{aligned}
C(z) &= \frac{1}{2} \sum_{(i,j) \in E} (1 - z_i z_j) \\
&= \frac{1}{2}(6 - z_2 z_0 - z_2 z_1 - z_2 z_3 - z_2 z_4 - z_0 z_1 - z_3 z_4) \\
&= \frac{1}{2}(6 + 1 + 1 + 1 - 1 - 1 + 1) = 4
\end{aligned}
$$

MaxCut: implementation in a quantum circuit

Cost function $\rightarrow$ Cost Hamiltonian

$$\hat{\mathcal{H}} = \frac{1}{2} \sum_{(i,j)\in E} (1 - \hat{\mathcal{Z}}_i \otimes \hat{\mathcal{Z}}_j)$$

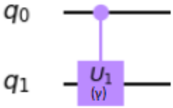
- $\hat{\mathcal{Z}}_i$ is Z-gate acting on qubit i
- Each qubit represents one node
- $z = \{z_1, z_2, \ldots, z_n\} \rightarrow z = |1\rangle\,|2\rangle \ldots |n\rangle$

Table of Contents

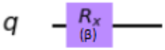
IBM's Qiskit Python module has been used to implement the QAOA algorithm in the following backends:

- qiskit's QASM-simulator with depolarizing noise on 1-qubit and 2-qubit gates.
- IBM's Yorktown 5 qubit Quantum Computer.

$$= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\gamma} \end{pmatrix}$$



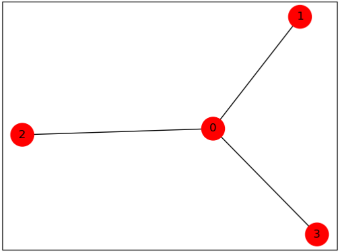
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\gamma} \end{pmatrix}$$



$$= \begin{pmatrix} \cos\frac{\beta}{2} & -i\sin\frac{\beta}{2} \\ -i\sin\frac{\beta}{2} & \cos\frac{\beta}{2} \end{pmatrix}$$

Quantum Circuit

- For the Star Graph with 4 nodes we have:



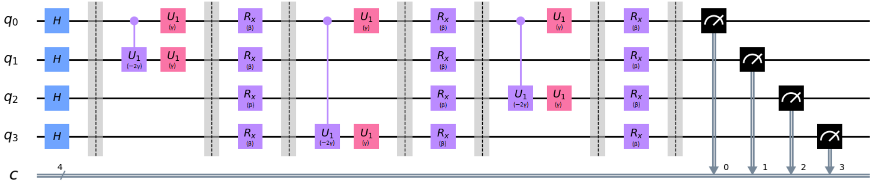
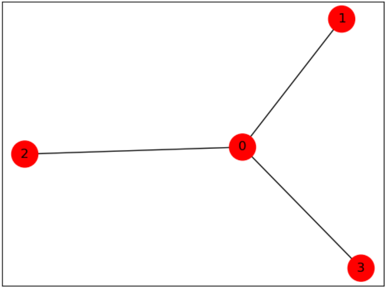
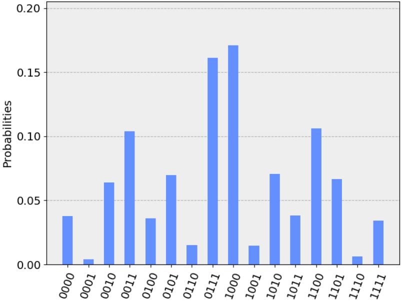


Table of Contents

Star Graph (Single layer)

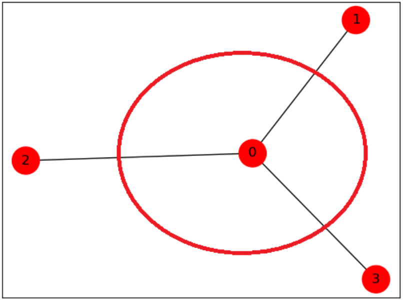
- Simulation run locally using qiskit's QASM-simulator without noise.

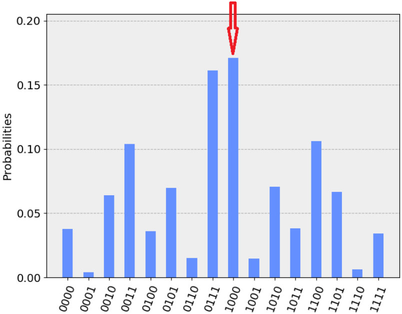




Star Graph (Single layer)

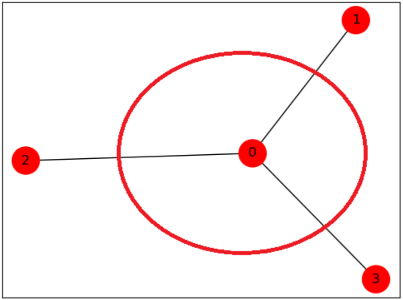
- Simulation run locally using qiskit's QASM-simulator without noise.

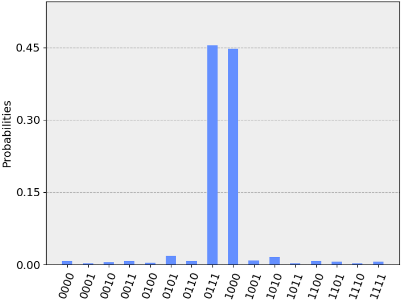




Star Graph (Triple layer)

- Simulation run locally using qiskit's QASM-simulator without noise.

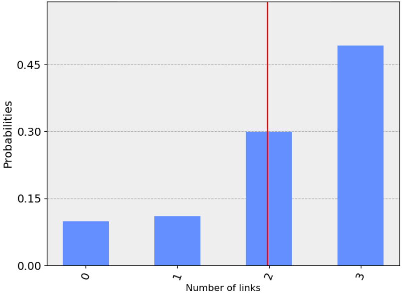




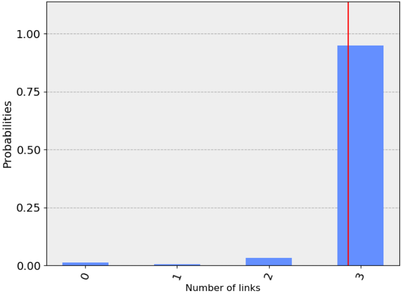
Cost function

Expressing the measurement outcomes as $z = \{-1, +1\}$, the associated cost function is:

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



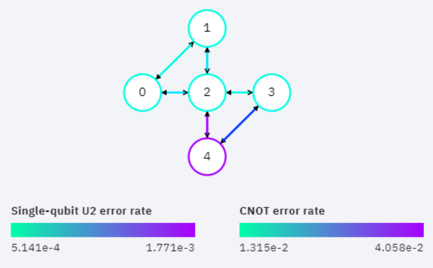
Single layer ($p = 1$)

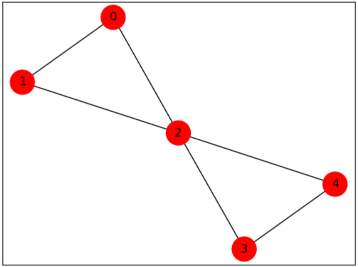


Triple layer ($p = 3$)

Butterfly Graph

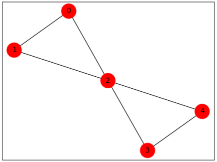
- The Butterfly Graph can be analyzed with IBM's Yorktown quantum computer.
- Yorktown hardware data: relaxation/dephasing time $\sim 25 - 80\mu s$, readout error ~ 0.02

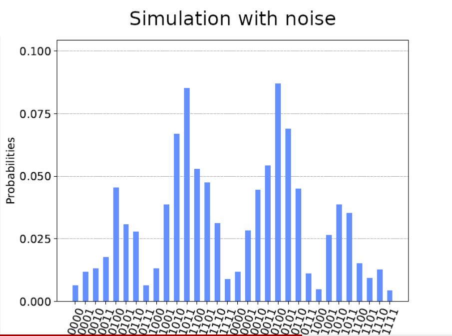


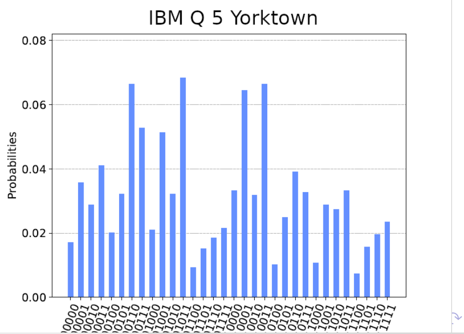


Butterfly Graph

- The Butterfly Graph can be analyzed with IBM's Yorktown quantum computer.
- Yorktown hardware data: relaxation/dephasing time $\sim 25 - 80\mu s$, readout error ~ 0.02
- We have also analyzed it with qiskit's QASM-simulator.

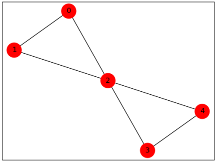


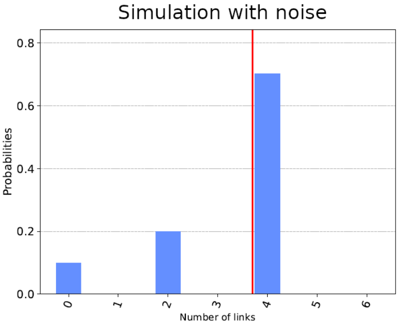


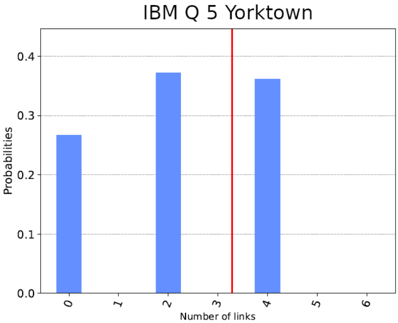


Butterfly Graph

- The Butterfly Graph can be analyzed with IBM's Yorktown quantum computer.
- Yorktown hardware data: relaxation/dephasing time $\sim 25 - 80\mu s$, readout error ~ 0.02
- We have also analyzed it with qiskit's QASM-simulator.

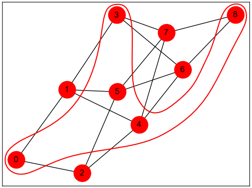


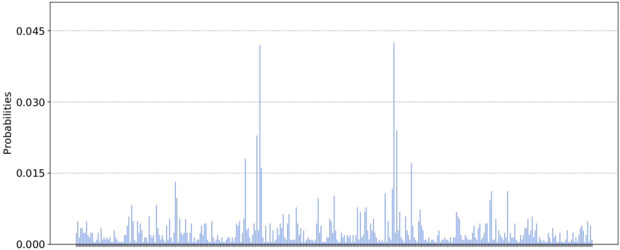




V9E15 Graph

- V9E15: 9 vertices, 15 edges
- One solution: all edges cut!





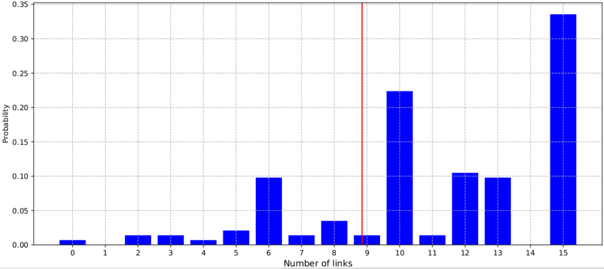


Table of Contents

Conclusion

- QAOA works! And even better for more layers, as long as there is no noise
- We are limited by the classical optimizer:
    - it takes 100-200 calls of the circuit to converge to optimal (β, γ) (slightly more in case of execution on hardware)
    - over 50% of the time is spent in the DE-routine (in case of simulation)
- When running on hardware we are limited by the quantum-classical feedback loop, which resets our position in the IBMQ queue

Future work

- Test different classical optimizers and run the code on a faster computer.
- Include readout errors and relaxation times in our simulated noise model.
- Analyze larger graphs on larger quantum computers, such as IBM's Melbourne 15 qubit QC

Thank you very much for your attention!

Any question?