QAOAToolkit: Bringing Quantum Optimization to the End User

T Anandakkoomar, Patrick Rebentrost, Trevor E. Carlson
National University of Singapore
https://github.com/nus-comparch/hamiltonian_engine

1 Introduction

- Noisy Intermediate Scale Quantum (NISQ) systems represent the current state of the art in quantum machines that can hold 50 - 100 qubits [1] and is limited by noise.
- Current NISQ Systems uses superconductors, trapped ions and photonics to simulate a qubit (IBM, Xanadu and Microsoft)
- NISQ systems have significant limitations that prevent us from achieving fault-tolerant quantum computers.
- Farhi's Quantum Approximate Optimization Algorithm [2] is one of the best candidate for current and future NISQ architectures since it requires both a quantum system and a classical computer to get results (hybrid heuristic).
- Quantum Alternating Operator Anstaz is an extended version of the algorithm which covers a wider range of problems and allows for more efficient methods to find the close to optimal answer by modifying some parts of the Hamiltonian

2 Motivation

Making QAOA approachable for non-experts in quantum computing

- Quantum computing can be hard for the uninitiated. However, QAOA is a promising heuristic.
- There needs to be a bridge between classical programmers and the quantum world.
- **QAOAToolkit** is a framework that aims to allow classical programmers to understand quantum optimization methods better using **Object Oriented Programming**.
- **Key Features :**
  - Abstraction of QAOA circuit generation
  - Predefined examples with minimal set-up
  - Plug and play concept to allow for flexibility

1.1 Quantum Alternating Operator Ansatz

- Works by alternating between a cost-function based cost/phase Hamiltonian and a mixer Hamiltonian.
- An objective function can be described as and exponentiated into a **Phase Hamiltonian, C**:

$$C(z) = \sum_{\alpha=1}^{m} C_\alpha(z)$$

$$U(C,\gamma) = e^{-i\gamma C} = \prod_{\alpha}^{m} e^{-i\gamma C_\alpha}$$

- In order to produce dynamicity between states a **Mixer Hamiltonian, B** is required:

$$B = \sum_{j=1}^{n} \sigma_j^x$$

$$U(B,\beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x}$$

- By combining both Hamiltonians:

$$|\psi'(\gamma,\beta)\rangle = U(B,\beta)U(C,\gamma)\,|\psi\rangle$$

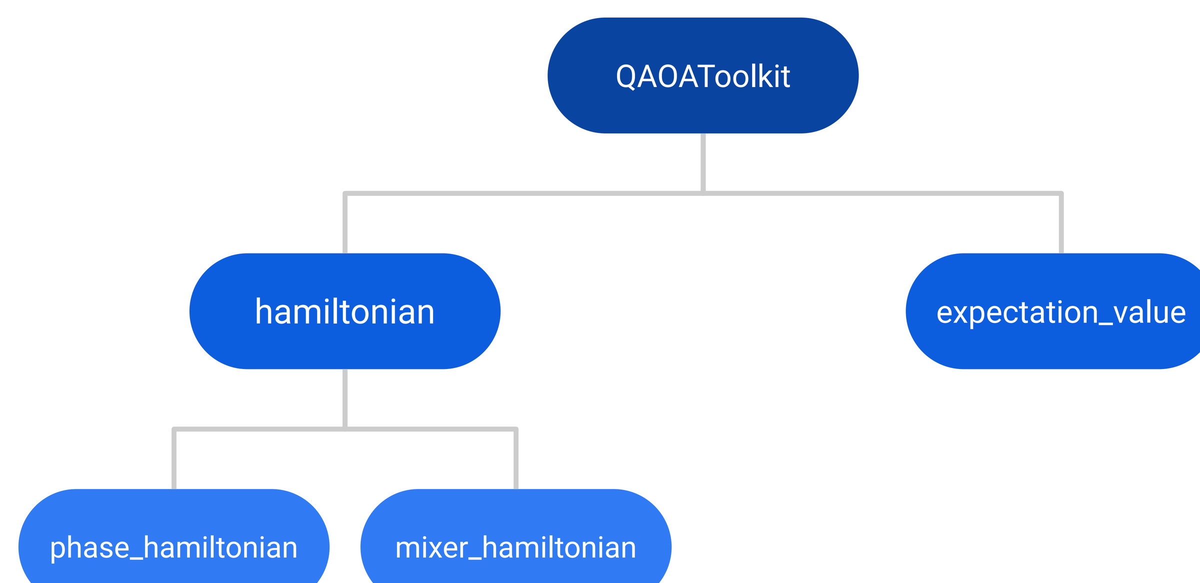


*Fig. 1 - **Class Diagram** - Chart showing the three most basic classes in the toolkit. Both phase_hamiltonian and mixer_hamiltonian classes share a parent so that arithmetic operations can be done on the hamiltonians and made easier for circuit generation.*

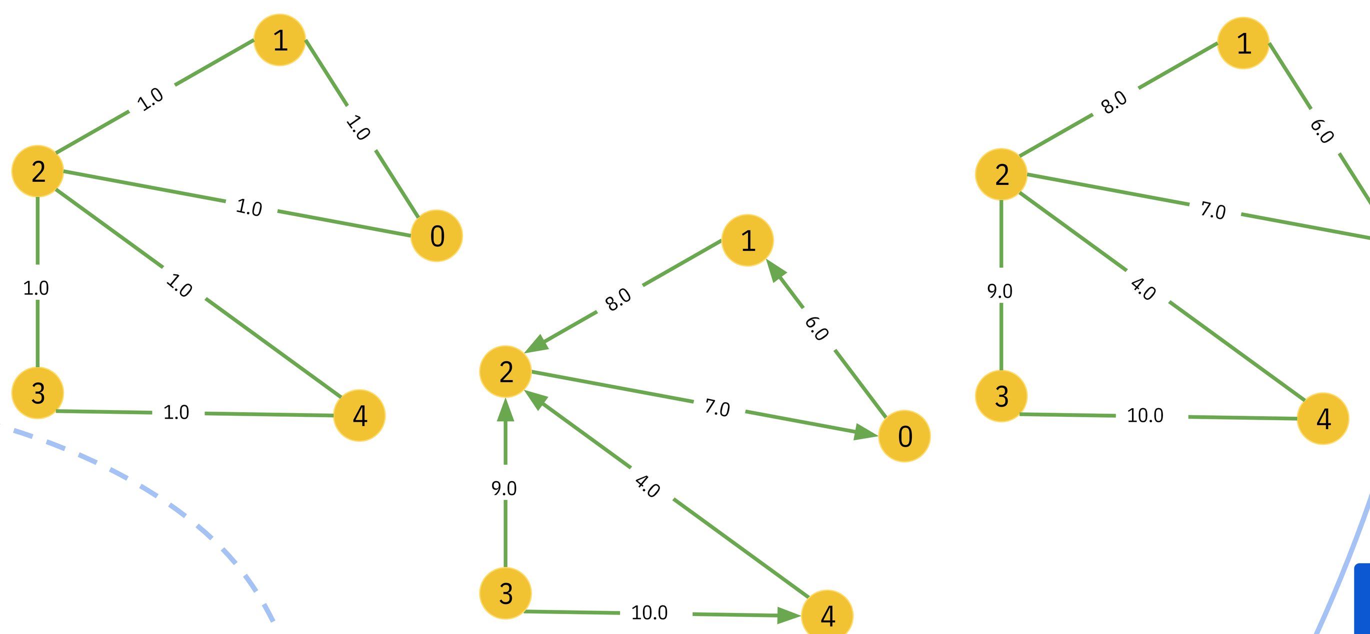


*Fig. 2 - **Types of graphs supported by the toolkit** - The QAOAToolkit is able to generate hamiltonians and circuits for the types of graph shown above; unweighted-undirected, weighted-undirected and weighted-directed.*

3 How it Works

- Consists of **3 major classes**:
  - Phase hamiltonian
  - Mixer hamiltonian
  - Expectation value
- Provides a high-level Python API for the QAOA heuristic [3] for those not experts in quantum computing using simple OOP concepts to better understand the workings of QAOA.
- **Hamiltonian classes**: Oversees the formation of the unitary operators and implementation of the unitaries into a quantum circuit.
- **Expectation value class**: Deals with the post processing of the results from the quantum computer/simulator.
- The split in classes is to give users flexibility when using the framework if they only require the use of a particular class/process.
- **Predefined examples:** Two problems are currently available to aid users to quickly deploy a quantum circuit.
- **Skeletor class:** Where users are given flexibility to change the objective function, optimizer and the device.

3.4 Skeletor - Plug and Play

- Maximize automation of the QAOA process, while providing flexibility.
- Ability to choose:
  - Objective function
  - Type of optimizer:
    - Scipy.optimize library
    - Scikit.optimize
    - Google's Quantum Tensorflow
- Allows users to compare different approaches for QAOA; easier to set-up experiments.

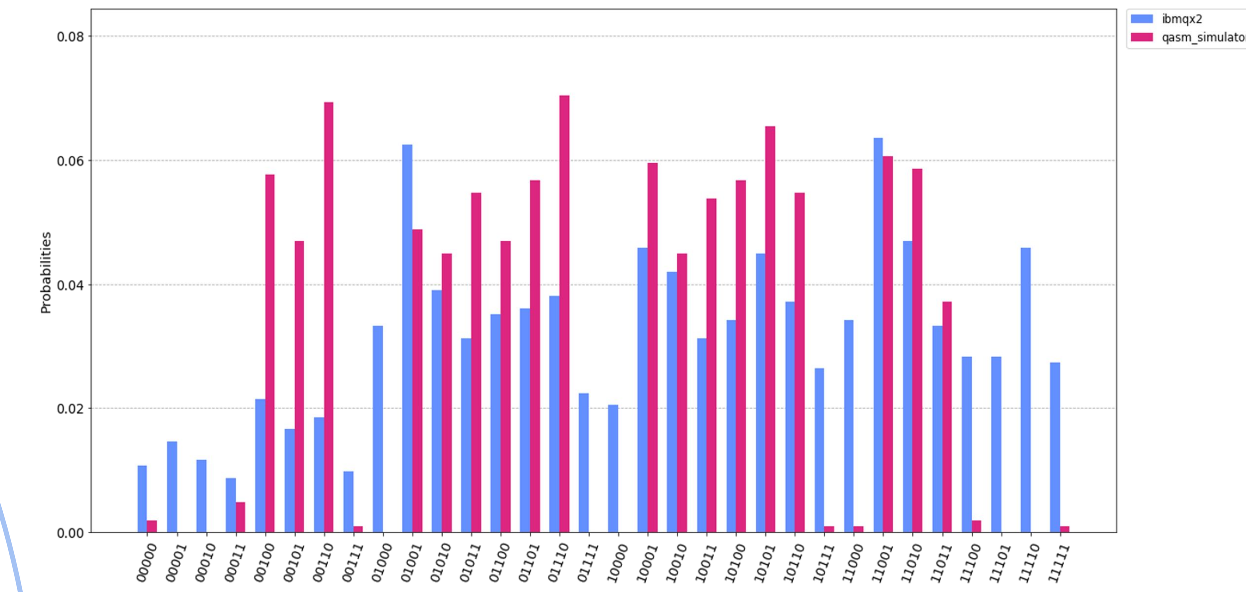


*Fig . 6 - **Comparing two results** - By allowing for an OOP concept, multiple instances of the skeletor objects can be created with ease and be used on different devices: a quantum simulator or a quantum computer.*

3.3 Predefined Examples

- To aid classical programmers to transition into quantum programming.
- Amalgamation of all classes to provide a higher layer of abstraction.
- Currently contains two problem classes: Max-cut & Max Independent Set.
- Able to handle directed and weighted graphs.
- User can choose a optimizer of their choice to get the close-to-optimal results.

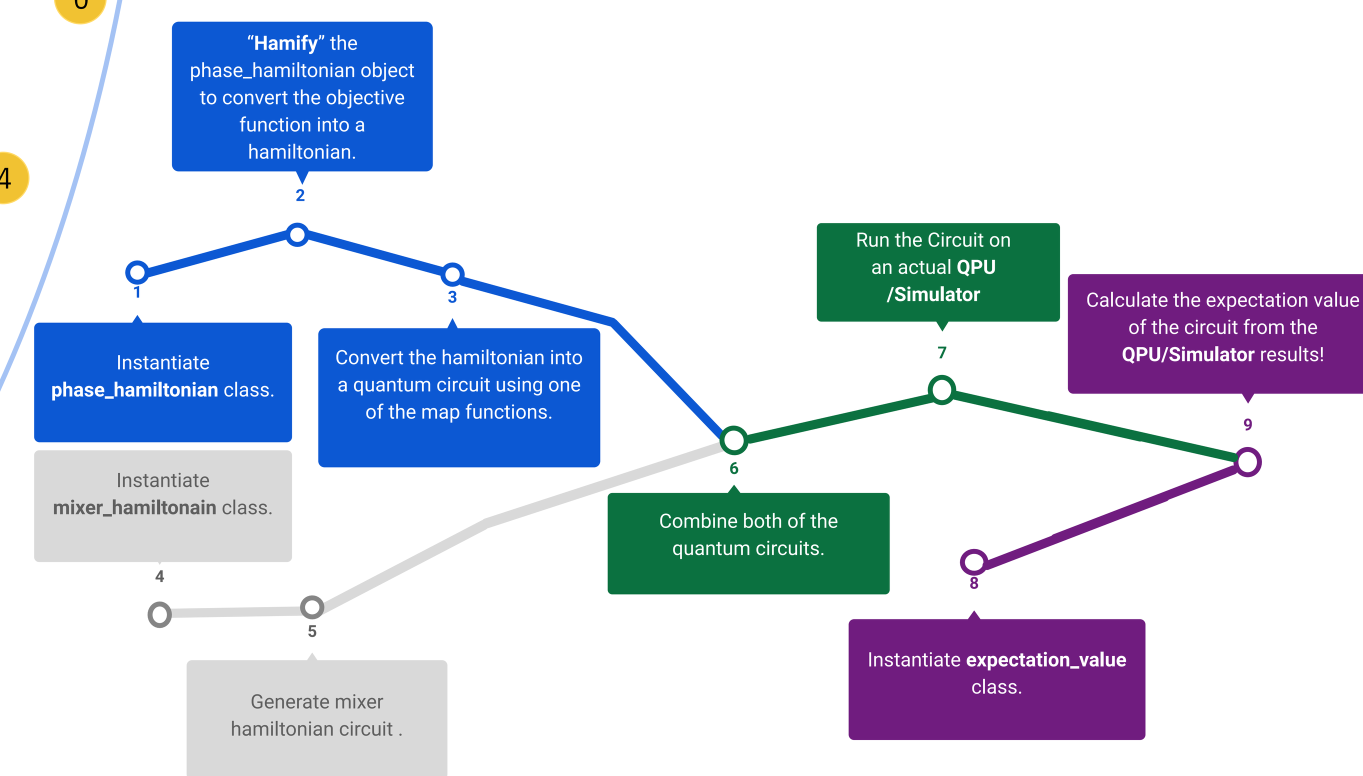


*Fig. 5 - **Flow diagram** - Steps taken to complete one iteration of the QAOA heuristic, after step 9, the data is sent to the classical optimizer to find the next best possible values of γ and β to get the close-to-optimal results.*

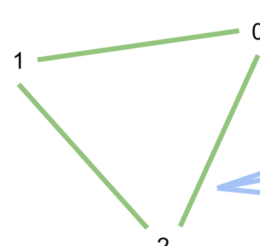
3.2 Expectation Value

- Simple class that makes sense of the results from the quantum device or simulator.
- Calculates the expectation value of the circuit using the equation :

$$\langle\psi'(\boldsymbol{\gamma},\boldsymbol{\beta})\rangle = |\alpha_{x_1}|^2C(x_1) + |\alpha_{x_1}|^2C(x_2) + |\alpha_{x_3}|^2C(x_3) + |\alpha_{x_4}|^2C(x_4)$$

3.1 Hamiltonian

- 2 Hamiltonian classes :
  - `Phase_hamiltonian`
  - `Mixer_hamiltonian`
- **Phase_hamiltonian**
  - Converts classical objective function into QAOA phase hamiltonian expression.
  - Converts the Hamiltonian expression into QAOA circuit using three different functions.
    - `perQubitMap : Maps each variable to an individual qubit.`
    - `perEdgeMap  : Maps each vertex in graph to a qubit.`
    - `perDitMap   : Maps each vertex to k-qubits.`
- **Mixer_hamiltonian**
  - Two functions available to generate qubits, unlike the phase_hamiltonian class which requires a objective function, mixer hamiltonian do not.
    - `generalXMixer    : applies Rx gate on each qubit.`
    - `controlledXMixer : applies a Controlled Rx gates based on the neighbouring vertices of the target vertex.`



perQubitMap()

perDitMap ()

perEdgeMap()

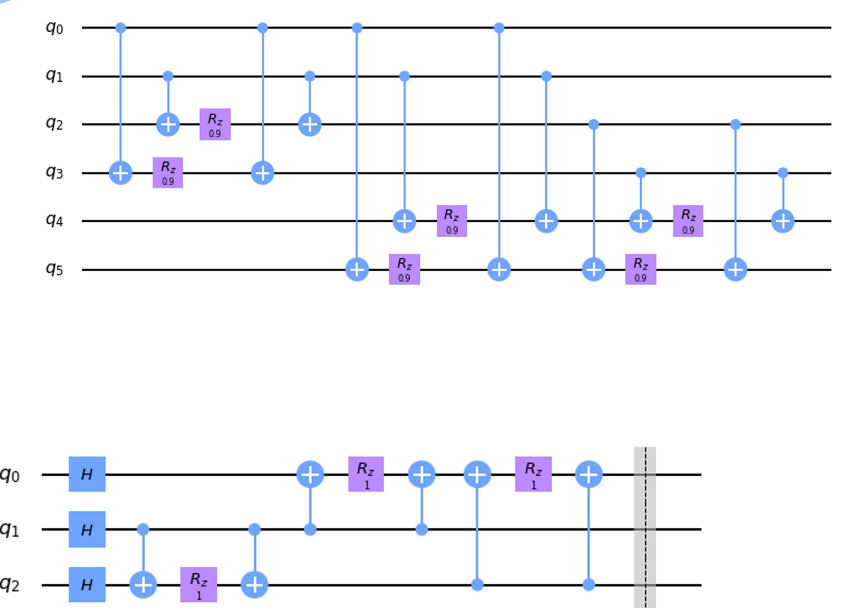


*Fig. 3 - **Different circuits can be generated**- perDitMap and perEdgeMap function both require the network graph to generate the circuit. While the perQubitMap function does not, and can be used for forming QUBO Hamiltonians.*

*Fig. 4 - **Two Different Methods for generating mixer hamiltonian circuits**- controlledMixer() improves the search space by only allowing dynamicity within the feasible search space unlike the generalXMixer() which moves between all Hilbert space [2].*

References

[1] John Preskill. Quantum computing in the nisq era and beyond. Quantum, 2:79, Aug 2018.
[2] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.
[3] Stuart Hadfield. Quantum algorithms for scientific computing and approximate optimization, 2018.

Acknowledgements

We would like to thank Entropica Labs for helping us to learn more about QAOA and quantum computing.