http://pubsonline.informs.org/journal/ijds

2024 INFORMS TUTORIALS
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 2694-4022, EISSN 2694-4030

Submitted to *2024 INFORMS TutORials*

Five Starter Problems: Solving Quadratic Unconstrained Binary Optimization Models on Quantum Computers

Arul Rhik Mazumder
Quantum Technologies Group, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA, arulm@andrew.cmu.edu

Sridhar Tayur
Quantum Technologies Group, Tepper School of Business, Carnegie Mellon University, Pittsburgh, PA 15213, USA, stayur@cmu.edu

Abstract. This tutorial offers a quick hands-on introduction to solving Quadratic Unconstrained Binary Optimization (QUBO) problems on currently available quantum computers. We cover both IBM and D-Wave machines: IBM utilizes a gate/circuit architecture, and D-Wave is a quantum annealer. We provide examples of three canonical problems and two models from practical applications . An associated GitHub repository provides the implementations in five companion notebooks. In addition to undergraduate and graduate students in computationally intensive disciplines, this article also aims to reach working industry professionals seeking to explore the potential of near-term quantum applications.

Key words: Quadratic Unconstrained Binary Optimization; Quantum Approximate Optimization Algorithm; Quantum Annealing; Simulated Annealing

1. Introduction

Quantum Computing is one of the most promising new technologies of the 21st century. Due to the nascency of the hardware, however, many of the envisioned speedups remain largely theoretical. Despite the limitations of available quantum hardware, there is significant interest in exploring them. While several articles and books (see (30) for example) adequately cover the principles of quantum computing, they typically stop short of providing a road map to access quantum hardware and numerically solve problems of interest. This tutorial is aimed at providing quick access to both IBM and D-Wave machines, thereby covering two types of quantum algorithms, one based on gate/circuit model (34) and the other on quantum annealing (27), with *Notebooks* containing associated software. We hope that our tutorial can serve as a companion to the available books and articles.

We first briefly introduce Quadratic Unconstrained Binary Optimization (QUBO) and Ising models. We then show how they capture three canonical combinatorial optimization problems: Number Partitioning (28), Max-Cut (9), and Minimum Vertex Cover (7). We next model two practical problems: Order Partitioning (for A/B testing at a hedge fund portfolio manager) and de novo discovery of driver genes in the

field of cancer genomics (1). We then provide step-by-step instructions [1] to solve the QUBOs across two frameworks: gate/circuit model (IBM hardware (32)) and quantum annealing (D-Wave hardware (6)).

2. Quadratic Unconstrained Binary Optimization and Ising models: The Bare Minimum

2.1. Quadratic Unconstrained Binary Optimization (QUBO) Models

Quadratic Unconstrained Binary Optimization (QUBO) models can be used to represent a variety of problems (5) (18). The general matrix form of a QUBO is:

$$\min_{\mathbf{x}\in\{0,1\}^n} [\mathbf{x}^T\mathbf{Q}\mathbf{x} + c].$$

Here $\mathbf{x}$ represents a binary solution vector of size n, $\mathbf{Q}$ is an upper triangular matrix, and c is an arbitrary constant. (Note that c makes no difference to the optimal solution.) After multiplying out the matrix form, the resulting equation is a quadratic unconstrained binary optimization model (QUBO):

$$\min_{x\in\{0,1\}^n} [\sum_{i,j} Q_{ij}x_ix_j + \sum_i Q_{ii}x_i^2 + c].$$

Since $\mathbf{x}$ is a binary vector, $x_i^2 = x_i$, we obtain the following triangular form:

$$\min_{x\in\{0,1\}^n} [\sum_{i<j} Q_{ij}x_ix_j + \sum_i Q_{ii}x_i + c].$$

QUBO models are a natural gateway to quantum computing due to their link with computer science and their mathematical equivalence to the Ising model (26) in physics. Note that a QUBO can be mapped to an Ising model using the transformation $\sigma_i = 2x_i - 1$, and conversely the conversion from Ising to QUBO is $x_i = \frac{\sigma_i+1}{2}$.

2.2. Ising Models

Ising Models are mathematical models used in statistical mechanics to model ferromagnetism. The generic form of an Ising model is:

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

In the model above σ corresponds to the the state of a collection of n particles, and each σ_i correspond to the eigenstate of the i particles of the system. The eigenstate of each σ_i corresponds to the Ising spin of a particle $\{-1, +1\}$. H is the Hamiltonian or energy function of the system which maps eigenstates to their corresponding eigenenergies. The remaining coefficients are J_{ij} which is the interaction between particles i, j based on their spins σ_i, σ_j, and h_i is the external bias applied at each particles σ_i. All in all, the Ising Model derives the energy of configuration $H(\sigma)$ based on the 2-body interactions $J_{ij}\sigma_i\sigma_j$ and

[1] For completeness (and for ability to compare with a classical benchmark), we also include Simulated Annealing (SA) in our tutorial.

the biases at each particle $h_i\sigma_i$. This model is relevant to quantum computing, as it can be easily applied to understanding the physical quantum annealing process on real D-Wave Quantum Computers. On real physical qubits, the J_{ij} coefficients are simulated by couplers which entangle their eigenstates, and the h_i coefficients are magnetic biases. This real-life applicability is why Ising models (or equivalently QUBO models) are the ideal way to frame optimization problems for quantum computers.

3. Three Canonical Problems

We have chosen three well-known canonical problems to formulate as QUBO models and solve with quantum algorithms. These three target problems: Number Partitioning, Max-Cut, and Minimum Vertex Cover are part of Karp's 21 NP-Complete Problems and thus are well known and studied in the field of theoretical computer science. It is because of this prevalence, that these problems were selected as readers can reference other literature on different formulations and techniques to benchmark the current approaches in this paper against. Furthermore, by definition of NP-complete problems, there are no exact classical polynomial-time solutions to these problems making them ideal targets for quantum speedup.

3.1. Number Partitioning

The Number Partitioning problem goes as follows: Given a set S of positive integer values $\{s_1, s_2, s_3 ... s_n\}$, partition S into two sets A and $S \setminus A$ such that:

$$d = |\sum_{s_i \in A} s_i - \sum_{s_j \in S \setminus A} s_j|$$

is minimized. In simple words, the goal is to partition the set into two subsets where the sums are as close in value to each other as possible. If $d = 0$, a perfect partition needs to be found; otherwise, d should be made as small as possible.

This objective function can be expressed as a QUBO by using the following binary variable: $x_i = 1$ indicates that s_i belongs to A and else s_i in $S \setminus A$. The sum of elements in A is $\sum_{i=1}^{n} s_i x_i$ and the sum of elements in $S \setminus A$ is $c - \sum_{i=1}^{n} s_i x_i$, where c is the sum of elements of S, a constant. Thus, the difference in the sums is:

$$d = c - 2\sum_{i=1}^{n} s_i x_i.$$

This difference is minimized by minimizing the QUBO:

$$d^2 = (c - 2\sum_{i=1}^{n} s_i x_i)^2 = c^2 + 4\mathbf{x}^T \mathbf{Q} \mathbf{x}$$

with q_{ij} in $\mathbf{Q}$ defined below:

$$q_{ij} = \begin{cases} s_i(s_i - c) & \text{if } i = j \\ s_i s_j & \text{if } i \neq j \end{cases}$$

The Number Partitioning Problem and its formulations are well-known and heavily studied in computer science (2), (29).

3.2. Max Cut

The Max-Cut Problem goes as follows: Given an undirected graph G with vertex set V and edge set E, partition V into sets A and $V \setminus A$ such that the number of edges connecting nodes between these two sets is maximized. See Figure 2 for a graph with six nodes labeled $0, 1, 2, 3, 4, 5$ and eight edges.

Similar to Number Partitioning, the QUBO can be constructed by setting $x_i = 1$ if vertex i is in A and letting $x_i = 0$ if vertex i is in $V \setminus A$. The expression $x_i + x_j - 2x_ix_j$ identifies whether edge $(i, j) \in E$ is in the cut: $x_i + x_j - 2x_ix_j$ is 1 if and only if exactly one of x_i or x_j is 1 (and the other is 0). The objective function is:

$$\max[\sum_{(i,j)\in E} x_i + x_j - 2x_ix_j]$$

Since QUBOs are typically framed as minimization problems, the Max-Cut QUBO is:

$$\min[\sum_{(i,j)\in E} 2x_ix_j - (x_i + x_j)]$$

Note that this can be expressed as

$$\min \mathbf{y} = \mathbf{x}^T \mathbf{Q}\mathbf{x}$$

with a matrix $\mathbf{Q}$ where the linear terms represent the diagonal and the quadratic terms represent the off-diagonal elements. The Maximum Cut Problem is one the quintessential combinatorial optimization problems (9) that has been studied in a variety of contexts, notably being the target of different approximation algorithms such as the famous Goemans-Williamson algorithm which inspired the Semidefinite Programming Paradigm (16) and the original QAOA algorithm (14).

3.3. Minimum Vertex Cover

As the name implies, the Minimum Vertex Cover problem seeks to find the minimum vertex cover of an undirected graph G with vertex set V and edge set E. A vertex cover is a subset of vertices such that each edge $(i, j) \in E$ shares an endpoint with at least one vertex in the subset.

The Minimum Vertex Cover problem has two features:

1. the number of vertices in the vertex cover must be minimized, and

2. the edges must be covered by the vertices. (All edges must share at least one vertex from the subset of vertices.)

If the inclusion of each vertex in the vertex cover is denoted with a binary value (1 if it is in the cover, and 0 if it is not), the minimal vertex constraint can be expressed as:

$$Q_1 = \sum_{i\in V} x_i$$

Similar to Max-Cut, the covering criterion is expressed with constraint (for all $(i, j) \in E$):

$$x_i + x_j \geq 1$$

This can be represented with the use of a penalty factor P:

$$Q_2 = P \cdot (\sum_{(i,j)\in E} (1 - x_i - x_j + x_i x_j))$$

Thus, the QUBO for Minimum Vertex Cover is to minimize:

$$\sum_{i\in V} x_i + P \cdot (\sum_{(i,j)\in E} (1 - x_i - x_j + x_i x_j))$$

The value P determines how strictly the covering criteria must be enforced. A higher P will ensure the vertex set covers the graph, at the expense of guaranteeing finding the minimal set. On the other hand, a lower P will have fewer elements in the vertex set but the set may not cover the graph.

The Minimum Vertex Cover problem is a well-known classical NP-Complete problem with many useful reductions. As a result, it is a well studied and documented problem (43), (46).

4. Two Practical Problems

Below we introduce two practical problems that are novel. As a result, there is no pre-existing literature on these problem formulations and their applications. The Ordering Partitioning problem was developed for the first time in this paper; thus, no reference materials discuss formulations or techniques. One useful related problem that has been well-studied is the Portfolio Optimization problem, and insights into this problem may apply to the Order Partitioning problem as well (25), (15). The Cancer Genomics problem identifies altered cancer pathways using mutation from the TCGA acute myeloid leukemia (AML) (24) and is NP-Hard. The problem formulation technique used in this paper was taken from one of the author's previous works (1).

4.1. Order Partitioning

Order Partitioning is used for A/B testing of various investment strategies (proposed by researchers at Hedge Funds). Also known as split testing, A/B makes variable-by-variable modifications and allows firms to optimize their strategies while avoiding the risks of testing at a larger scale. Order Partitioning explicitly reduces from Number Partitioning and thus is an NP-complete. This class of problems by definition has no good polynomial-time algorithms and are viable candidates for quantum speedup (44).

The Order Partitioning problem goes as follows: We are given a set of n stocks each with an amount q_j dollars totaling T dollars ($\sum_{j=1}^{n} q_j = T$). There are m risk factors. Let p_{ij} be the risk exposure to factor i for stock j. The objective is to partition the stocks $j = 1 \dots n$ into sets A and B such that:

1. the sizes are equal ($\sum_{j\in A} q_j = \sum_{i\in B} q_j = \frac{T}{2}$) and
2. the risks are equal (for each factor $i = 1 \dots m$: $\sum_{j\in A} p_{ij} = \sum_{j\in B} p_{ij}$.)

If exact equality is not possible, then we want the absolute difference to be minimized:

1. $\min |\sum_{j\in A} q_j - \sum_{j\in B} q_j|$ and

2. for each factor $i = 1 \dots m$: $\min |\sum_{j \in A} p_{ij} - \sum_{j \in B} p_{ij}|$.

Since Objective 1 for each case is the same as the Number Partitioning problem, we can reuse the QUBO derived earlier (replacing s with q, and noting that x_j is binary).

$$Q_1 = (T - 2\sum_{j=1}^{n} q_j x_j)^2.$$

To satisfy Objective 2 for each case, we aim to reduce the absolute risk, thus we would first like to reduce the absolute risk over each stock before minimizing its overall stocks. Considering the $\{-1, 1\}$ Ising spin variable, the minimum absolute risk is 0. This case is similar to the number partitioning problem where we absolute risk is minimized by minimizing the square of the sum.

$$\min(\sum_{j=1}^{n} p_{ij}\sigma_j)^2$$

Considering the minimum risk across all stocks we get the following final expression to minimize across risk factors in terms of Ising spin variables

$$\min \sum_{i=1}^{m} (\sum_{j=1}^{n} p_{ij}\sigma_j)^2$$

Converting back to QUBO variables and weighting with penalty factor P we get:

$$Q_2 = P\sum_{i=1}^{m} (\sum_{j=1}^{n} p_{ij}(2x_j - 1))^2$$

The complete QUBO for Order Partitioning is:

$$Q = (T - 2\sum_{j=1}^{n} q_j x_j)^2 + P\sum_{i=1}^{m} (\sum_{j=1}^{n} p_{ij}(2x_j - 1))^2$$

Note that P depends on how strictly each constraint is to be enforced.

4.2. Cancer Genomics

The second application is de novo identification of altered cancer pathways from shared mutations and gene exclusivity. A cancer pathway is an identified sequence of genes that is disrupted by the cancer. Its identification is useful in understanding the disease and developing treatments. This problem non-trivially reduces from Independent Set (which is well known to reduce from Vertex Cover (39)). Thus this practical problem is NP-complete and is a suitable candidate for quantum speedup.

The patients and their corresponding gene mutations are first modeled as the incidence matrix B of a hypergraph with each vertex g_i representing a gene, and each patient P_i represents the hyperedge. Note that this incidence matrix is of the form

$$B = \begin{bmatrix} b_{11} & b_{12} \dots b_{1m} \\ b_{21} & b_{22} \dots b_{2m} \\ \vdots & \vdots \ddots \vdots \\ b_{n1} & b_{n2} \dots b_{nm} \end{bmatrix}$$

each of the m columns represents each patient P_i mutation gene list, and each of the n rows represents the presence of gene g_j in a patient's mutated gene list. For example, if gene g_i is mutated for patient P_j then $b_{ij} = 1$ else $b_{ij} = 0$. Using this incidence matrix, we can derive the graph Laplacian as follows:

$$L^+ = BB^T$$

There are two key combinatorial criteria essential to identifying driver mutations:

- Coverage: We want to identify which genes are most prevalent among cancer patients. If a gene is shared between many cancer patients, it is more likely that the gene is a driver gene.
- Exclusivity: If there is already an identified cancer gene in the patient's gene list, it is less likely for there to be another. This is not a hard rule and is sometimes violated.

Based on the criterion above, we obtain the following two matrices, by decomposing the graph Laplacian L as

$$L^+ = \mathbf{D} + \mathbf{A}$$

- Degree Matrix $\mathbf{D}$: This diagonal matrix corresponds to the coverage criterion. Each d_{ij} index where $i = j$ represents the number of patients that are affected by gene i. All other entries are 0 and this attribute should be maximized.
- Adjacency Matrix $\mathbf{A}$: This adjacency matrix corresponds to the exclusivity criterion. Each a_{ij} with $i \neq j$ represents the number of patients affected by gene i and gene j. All other entries are 0 and this attribute should be minimized.

Let us begin with attempting to find just one pathway. We define a solution pathway as $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & ... & x_n \end{bmatrix}^T$ where for all $i \in \{1, 2, 3, \ldots, n\}$, x_i is a binary variable. If $x_i = 0$, then gene i is not present in the cancer pathway. If $x_i = 1$, the gene is present in the cancer pathway. The exclusivity term is $\mathbf{x}^T\mathbf{A}\mathbf{x}$ and the coverage term is $\mathbf{x}^T\mathbf{D}\mathbf{x}$. Since we want the exclusivity term to be minimized and the coverage term to be maximized, the QUBO to identify cancer pathways from a set of genes is:

$$\mathbf{x}^T\mathbf{A}\mathbf{x} - \alpha\mathbf{x}^T\mathbf{D}\mathbf{x}$$

where α is a penalty coefficient. Since coverage is more important than exclusivity, we weigh it more, and $\alpha \geq 1$. This QUBO can be written equivalently using the a_{ij} and d_{ij} values of A and D respectively as follows:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_ix_j - \alpha\sum_{i=1}^{n} d_ix_i.$$

This QUBO can be extended to identify multiple cancer pathways in a single run (23). Note that this is just one possible formulation of formulating the cancer pathway problem, however other approaches do exist. See for example (42), (45).

5. Quantum Computing Background

We provide a very succinct introduction to quantum computing. Readers are advised to use this section to get a fundamental understanding of how quantum computing works, however, it is not essential to know every little detail. In the same way, programmers don't need to understand how every bit is manipulated when writing an algorithm, quantum programmers don't need to visualize all qubit operations. That being said it will be useful to read this section or similar materials to get a better understanding of the subject matter. Also, quantum computing is an expansive and growing field and there exist many more interesting nuanced concepts beyond the ones discussed here, however, this context is sufficient to understand the rest of the paper.

At the moment there exist two main paradigms of quantum computing: gate-based quantum computing and adiabatic quantum computing. Both models theoretically have the same computational power but work in fundamentally different ways. Each model has one core algorithm that we will thoroughly investigate and implement. The gate-based model relies on the Quantum Approximate Optimization Algorithm to solve combinatorial optimization problems. The primary implementation of adiabatic quantum computing is done as Quantum Annealing. Both algorithms will be discussed in further detail in the following section.

5.1. Qubits

Similar to how the base unit of classical computing is a bit, the fundamental unit of quantum computing is the quantum bit or qubit. Unlike binary bits which can be measured in 0 or 1, qubits can be measured in states $|0\rangle$, $|1\rangle$ or a superposition of both states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

$|0\rangle$ and $|1\rangle$ are the measurable computational basis states representing vectors in a 2D Hilbert Space:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

The exact mathematics goes beyond the scope of the paper, however, a Hilbert Space can be understood as a special type of vector space.

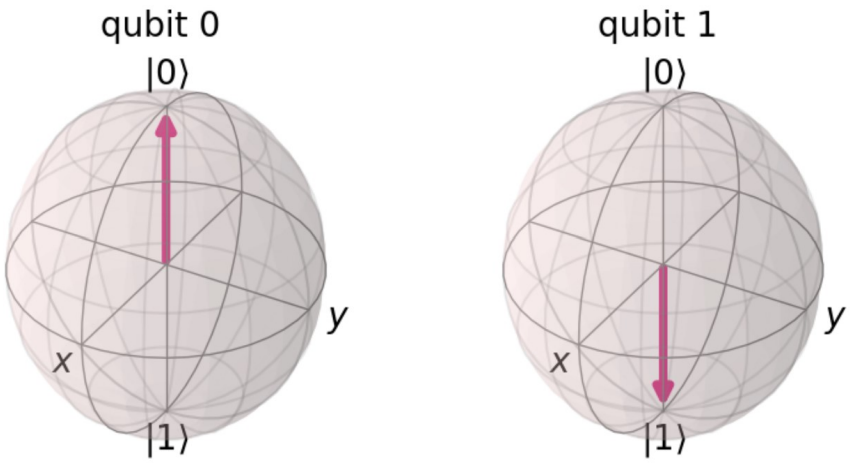


Figure 1 Bloch Sphere representations of $|0\rangle$ and $|1\rangle$ respectively

The x-axis and y-axis represent the real and imaginary parts of the probability amplitude respectively. the z-axis represents the probability difference between the $|0\rangle$ and $|1\rangle$ states.

Also, note that vectors are expressed as kets: $|\psi\rangle$. This notation is known as Dirac Notation and is commonly used in quantum computing.

Coefficients α and β are complex-valued probability amplitudes that when squared represent the probability of measuring each state. This idea is known as Born's Rule and from this we can determine the probability of measuring each basis state and show the fundamental idea in probability:

$$P(0) = |\alpha|^2, P(1) = |\beta|^2$$

$$|\alpha|^2 + |\beta|^2 = 1$$

Because of this ability for a qubit to probabilistically represent two states at once, a quantum register of n qubits could have a superposition of 2^n states at once, with each state occurring with a certain probability. Below is the uniform distribution of n qubits where $|i\rangle$ represents the ith bitstring:

$$\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$$

A classical register on the other hand would only be able to map one state. This ability for qubits to map multiple states is known as quantum parallelism and contributes to quantum speedup.

When multiple quantum states (each in different Hilbert Spaces $\mathcal{H}_i$) or registers are combined such as $|\psi_1\rangle \in \mathcal{H}_1$, $|\psi_2\rangle \in \mathcal{H}_2$,.., $|\psi_n\rangle \in \mathcal{H}_n$, the resulting system is the tensor or Kronecker product of the states: $|\psi_1\rangle \otimes |\psi_2\rangle \otimes ... \otimes |\psi_n\rangle = |\psi_1\psi_2...\psi_n\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \mathcal{H}_3 \otimes \ldots \otimes \mathcal{H}_n$. Below is an example of taking the tensor product of a $|0\rangle$ and $|1\rangle$:

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 0 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

5.2. Quantum Gates

Qubits are manipulated using unitary operators U. These operators are defined by 2 key properties:

1. They only map unit vectors to unit vectors.
2. They are adjoint (i.e. Given unitary operator U and its conjugate transpose $U^\dagger$, $UU^\dagger = U^\dagger U = I$)

These unitary operators are known as quantum gates. Note that these operators can be visualized as reflections and rotations of qubit vectors and unlike classical computing, are all reversible. Below is a brief list of single qubit quantum gates and functionality:

The trivial Identity gate I is often ignored when implementing circuits but is necessary for mathematics. It preserves the state and is the same as the identity matrix:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The Hademard gate H is used to convert basis states to a uniform superposition. This makes it equally likely to be measured in the $|0\rangle$ or $|1\rangle$ states, The matrix is shown below:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The Pauli Gates σ_X (or X), σ_Y (or Y), and σ_Z (or Z) are essential gates to rotate the quantum state vectors 180 across their corresponding axes. The three matrices are shown below:

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The Rotation Gates R_X, R_Y, and R_Z are similar to the Pauli Gates but implement parametrized rotations over their respective axes as shown in their matrices below:

$$R_x(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}$$

There also exists multi-qubit gates. The CX (also known as $CNOT$) and CZ read the input from a control qubit and then apply the corresponding σ_X or σ_Z operation. Their matrices are shown below:

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

There exist other single and multi-qubit quantum gates, however, these are the only ones essential to this tutorial. Similar to combining a system of qubits, multiple operators can be combined over multiple qubits using the tensor product. For example, for the simple Bell state circuits shown below from IBM Composer:

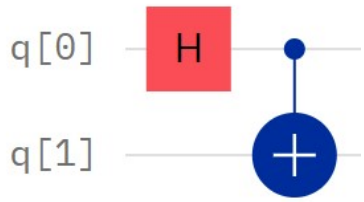


Figure 2 **First Bell state equivalent to $|\phi^+\rangle = \frac{|00\rangle+|11\rangle}{\sqrt{2}}$.**

The corresponding equation where $q[0] = |0\rangle$ and $q[1] = |0\rangle$ is:

$$[(CNOT) \times (I \otimes H)] \, |00\rangle$$

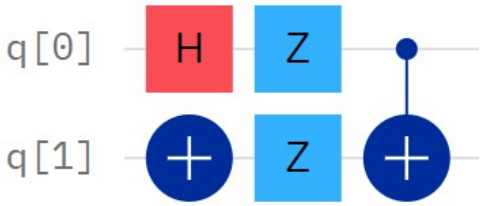


Figure 3 **Fourth Bell state equivalent to $|\Psi^-\rangle = \frac{|01\rangle-|10\rangle}{\sqrt{2}}$.**

The corresponding equation where $q[0] = |0\rangle$ and $q[1] = |0\rangle$ is:

$$[(CNOT) \times (Z \otimes Z) \times (X \otimes H)] \, |00\rangle$$

Note that although the circuits are read from left to right, it is written as a matrix expression from right to left due to matrix notation. This order is important due to the non-commutativity of matrix multiplication. While the matrices in each "column" are tensored from bottom to top, this can be changed due to the commutative property of tensor multiplication.

5.3. Entanglement

Note that two states $|\phi^+\rangle$ and $|\Psi^-\rangle$ are interesting due to being Bell States. Bell States are the simplest example of entangled states. Mathematically this means that they cannot be purely expressed as the tensor product of single qubits. For example an an arbitrary entangled state $|\psi\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$, we cannot write

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$$

with $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$.

More intuitively quantum entanglement is when multiple qubits are correlated such that their states are dependent on each other. Thus for example with the first Bell State: $|\phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, if the first qubit is measured as 0, the second one would also come up as 0. Similarly, if the first qubit is 1, the second one would also be 1. This property has been experimentally verified and offers quantum speedup potential.

5.4. Measurements

When a qubit is measured, its superposition is collapsed and deterministically measured in a $|0\rangle$ or $|1\rangle$ state. For example when measuring $|\psi\rangle$:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

It will come up as $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. Similarly for $|\phi\rangle$:

$$|\phi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$$

There is $\frac{1}{2}$ chance $|\phi\rangle$ is measured as $|0\rangle$ and $\frac{1}{2}$ chance $|\phi\rangle$ is measured as $|1\rangle$.

While the measurement gates are typically taken over any computational basis, they can be explicitly defined for any quantum state $|\psi\rangle$ through the following equation:

$$M_\psi = |\psi\rangle\,\langle\psi| \tag{1}$$

Note that $\langle\psi|$ is the bra of ψ and is the conjugate transpose of $|\psi\rangle$. Applying this for computational basis states, $|0\rangle$ and $|1\rangle$ we get:

$$M_0 = |0\rangle\,\langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$M_1 = |1\rangle\,\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

Based on the measurement gates M_m and quantum state $|\psi\rangle$. You can measure state m with probability:

$$P(m) = \langle\psi|\,M_m^\dagger M_m\,|\psi\rangle$$

5.5. NISQ

When it comes to the capabilities of current gate-based quantum computing, current technology is limited to the Noisy Intermediate Quantum Scale (NISQ) era. As demonstrated by the name NISQ processors are both limited by size and accuracy. The most powerful quantum computer, IBM's Osprey, only has 433 qubits limiting the scope of potential solvable problems. We are on track to have 1000+ qubit computers by 2025, however, current quantum capabilities are limited.

Furthermore, the hardware's quantum capabilities are extremely sensitive. As mentioned earlier qubits make up the core of quantum systems and their superposition and entanglement properties offer quantum speedup. Unfortunately, current qubits are noisy suffering from decoherence (loss of quantum properties) and quantum gate imperfections due to environmental influence. For these reasons only circuits with low quantum depth (few layers of quantum gates) have even moderate levels of accuracy. While there is active research on promising error correction schemes, they are not yet applicable to larger-scale devices preventing fault-tolerant computing.

Once both of these hurdles are overcome, post-NISQ era ten thousand qubit fault-tolerant computers can carry out extraordinary tasks like breaking RSA encryption using Shor's Algorithm. These fault-tolerant computers will also prove quantum supremacy; quantum computers solving tasks unfeasible for classical computers.

6. Algorithms

Three common algorithms are implemented: Quantum Approximate Optimization Algorithm (QAOA) (14), Quantum Annealing (QA) (11) and Simulated Annealing (SA) (21). In the following section, we will discuss the theory that guides each of these algorithms and provide a pseudocode representation.

6.1. Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) is a quantum-classical algorithm used to solve combinatorial optimization programs on gate-based quantum computers. At a higher level, it can be viewed as a trotterized version of quantum annealing. Because of its low circuit depth, it is one of the promising quantum applications that run on Noisy Intermediate Scale Quantum (NISQ) Hardware.

QAOA begins by mapping the optimization problem to an objective function with $z = (z_1, z_2..., z_N)$ and $z_i = \{0, 1\}$. Using this objective function, we calculate the expected value from a set of inputs, with lower expected values indicating higher solution quality. The algorithm begins to initialize random parameters $\{\beta\}_{i\geq 0}$ and $\{\gamma\}_{i\geq 0}$ in feed into the initial parametrized circuit.

6.1.1. Parametrized Quantum Circuits A parametrized quantum circuit is created by creating a layer of Hadamard gates and then using multiple alternating Mixer and cost Hamiltonians layers. The Hademard H gates are used to convert the basis states $|0\rangle$ a uniform superposition $|+\rangle$ to explore all configurations. The mixer Hamiltonian $e^{-i\beta H_M}$is created from the passed in $\{\beta\}_{i\geq 0}$. This operator is used to explore the search space more efficiently. The cost Hamiltonians $e^{-i\gamma H_C}$ evaluates the energy of different inputs and is created from the passed in $\{\gamma\}_{i\geq 0}$ and problem parameters, serving as a matrix version of the cost function $C(z)$. The number of times the mixer and cost operators are cycled depends on a parameter p. The entire quantum circuit is modeled in the equation below (41) with the circuit representation underneath. This circuit is parametrized since the vector θ with $2p$ parameters γ and β for cost and mixer operators are then adjusted during the classical optimization phase.

$$|\psi\rangle = (\prod_{i=1}^{p} e^{-\beta_i H_M} e^{-i\gamma_i H_C})|+\rangle$$

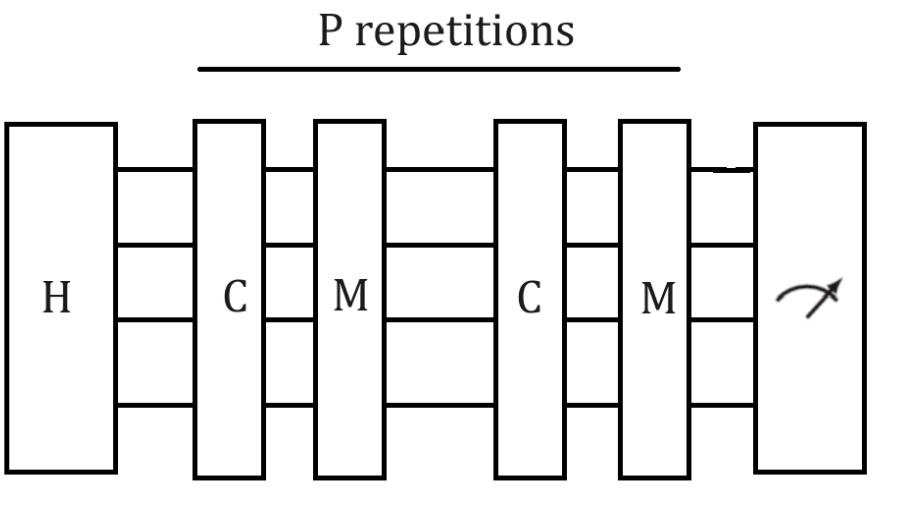


Figure 4 Generic QAOA Framework

6.1.2. Classical Optimization Given the parameters of the cost and mixer Hamiltonians, a classical optimization function is then run to minimize the expected energy of the problem Hamiltonian. The computation for this expected value is shown below:

$$E(\theta) = \langle \theta | H_C | \theta \rangle$$

Since the expectation function is non-convex, it is an active research topic on which optimizers are best, however, heuristics and gradient-based optimization methods are commonly employed to find the θ that minimizes $E(\theta)$. Once the optimization is done, the circuit is measured using the computational basis, and an approximate solution is obtained. The complete pseudocode for QAOA is shown below:

Algorithm 1 Quantum Approximate Optimization Algorithm

procedure QAOA(cost Hamiltonian H_c, Mixing Hamiltonian H_m, Number of steps p)
    Initialize state $|\psi\rangle = |+\rangle^n$ (all qubits in superposition)
    Initialize parameters $\vec{\beta}, \vec{\gamma} \in \mathbb{R}^p$ (randomly)
    for $i \in \{1, 2, \ldots p\}$ **do**
        for $j \in \{0, 1, 2, \ldots n-1\}$ **do**
            Apply Rx(β_i) to qubit j
        Apply Rz(γ_i) to all qubits
        Apply controlled-H_c operations based on problem structure
    Measure the state $|\psi\rangle$
    return Measurement outcome (solution)

6.2. Simulated Annealing

Simulated Annealing (SA) is a metaheuristic algorithm inspired by the concept of thermal annealing, where metals are slowly cooled to obtain improved physical properties (22).

Similar to QAOA, SA first maps the target problem to a cost function. The algorithm then initializes a random solution, and then iteratively improves upon this solution to find approximate global optimum solutions to the multivariate combinatorial or continuous cost.

At each iteration, it generates a neighbor by making small perturbations to the current solution and evaluates this neighbor solution energy using the cost function. If the neighbor has lower energy than the current solution, SA replaces the current solution with the neighbor. Unlike other metaheuristic functions, Simulated Annealing uses a unique Temperature Schedule and Cooling Process to determine whether to replace the current solution with an inferior neighboring solution (35).

6.2.1. Temperature Schedule

The probability of the solution being replaced with a higher neighbor depends on a temperature T initialized at the beginning of the algorithm. The exact acceptance probability function is:

$$p(f, x, x', T) = \exp(-\frac{f(x') - f(x)}{T})$$

This equation is taken from thermal annealing and controls the exploration process of the metaheuristic. As shown from the equation, the solution is likely to change if the temperature is high or the difference between the energies ΔE is minimized. The value of T is typically initialized to a large value to encourage more explorative behavior and decreases over time due to the Cooling Process. The biggest limitation of SA is its inability to explore solutions where there is a large ΔE (11).

6.2.2. Cooling Process

At each iteration, the temperature exponentially decreases by a constant cooling factor $0 < \alpha < 1$ as modeled in the equation below.

$$T_{new} = \alpha * T_{old}$$

As the temperature decreases, the algorithm is less likely to choose suboptimal neighbors and exploits the current solution more (8). The complete pseudocode for Simulated Annealing is shown in the figure below:

Algorithm 2 Simulated Annealing

procedure SIMULATED ANNEALING(Function f, Initial solution x, Probability Function p, Stopping criterion)
  $T \leftarrow T_{\text{initial}}$
  best solution $\leftarrow x$
  while $T > T_{\text{final}}$ **do**
    $x' \leftarrow$ Generate a random neighbor of x
    $\Delta E \leftarrow f(x') - f(x)$
    if $\Delta E < 0$ **then**
      $x \leftarrow x'$
      if $f(x') < f(\text{best solution})$ **then**
        best solution $\leftarrow x'$
    else
      if rand$(0,1) < e^{-\Delta E/T}$ **then**
        $x \leftarrow x'$
    $T \leftarrow \alpha \times T$
  return best solution

6.3. Quantum Annealing

Quantum Annealing (QA) is a quantum computing metaheuristic used to solve difficult optimization problems (12). It is inspired by Simulated Annealing and provides a theoretical advantage over classical algorithms due to its ability to leverage quantum mechanical principles, specifically quantum tunneling. Unlike many other quantum computing techniques, quantum annealing is very specialized towards optimization problems and can only be run on specialized quantum annealing platforms like D-Wave's quantum annealers.

In QA, the optimization problem is first converted to an objective function. This function is similar to the cost function mentioned in QAOA and is then mapped to a time-independent Hamiltonian H_f (33). This Hamiltonian describes the energy of the system, and its ground state can be mapped back to the solution to the optimization problem. A non-commutable time-dependent transverse field Hamiltonian H_t is also added with a traverse field coefficient $\Gamma(t)$. Thus the final Hamiltonian is:

$$H = H_f + \Gamma(t)H_t$$

6.3.1. Annealing Schedule The annealing schedule is managed by the earlier transverse field coefficient $\Gamma(t)$ which describes how the system changes over time. The initial $\Gamma(0)$ is taken to be very large but gradually decreases to 0 over the annealing process.

6.3.2. Time Evolution under Schrodinger Equation During annealing, the time evolution is derived using the time-dependent Schrodinger Equation:

$$i\hbar\frac{\partial\varphi}{\partial t} = (\Gamma(t)H_t + H_f)\varphi$$

In the equation above φ represents the quantum state at time t. According to the Adiabatic Theorem if the system evolves slowly enough $\Gamma(t) = 0$, and the ground state of H_f is isolated encoding the solution to the initial optimization problem which is then measured and returned as a bitstring (13).

6.3.3. Quantum Tunneling During time evolution, the quantum system could run into energy barriers where the differences in energy between the current state and neighboring are very large. Simulated Annealing often struggles with finding global optimum in these circumstances, due to its acceptance probability function, however, Quantum Annealing prevails due to Quantum Tunneling (37). Because of the wave-like properties, the Hamiltonians are encoded, and there is a non-zero probability that the system can tunnel through the energy barrier to find optimal solutions. This phenomenon is the main quantum mechanical advantage of QA.

Algorithm 3 Quantum Annealing

procedure QUANTUM ANNEALING(Hamiltonian H, Initial state $|\psi\rangle$, Traverse field function $\Gamma(t)$, Ground State)
  $t \leftarrow 0$
  best solution $\leftarrow |\psi\rangle$
  while not in ground state **do**
    $\Gamma_t \leftarrow \Gamma(t)$
    $H \leftarrow H_f + \Gamma_t H_t$
    Evolve state $|\psi\rangle$ under the Schrödinger equation for Δt using H
    $t \leftarrow t + \Delta t$
  best solution $\leftarrow$ Measure the ground state $|\psi\rangle$
  return best solution

7. Solving QUBOs on Quantum Computers

Three common algorithms are implemented: Quantum Approximate Optimization Algorithm (QAOA) (14), Quantum Annealing (QA) (11) and Simulated Annealing (SA) (21). QAOA is implemented in three different ways on an IBM Simulators: Vanilla QAOA (Notebook 1), OpenQAOA (Notebook 2) and Qiskit (Notebook 3). Note that these IBM simulators were chosen to run noise-free simulations on larger inputs. Specifically, IBM machines and simulators were chosen due to their ease of access. Larger IBM machines or machines from other organizations like Rigetti or Google required paid access, which we avoid for the

sake of this tutorial. The coding techniques work similarly across platforms due to OpenQAOA and Qiskit compatibility. Simulated Annealing (Notebook 4) and Quantum Annealing (Notebook 5) are implemented (on D-Wave) using the Annealing API from Ocean SDK. For brevity, in each of the five subsections, we detail only one of the five problems for each algorithm, thereby covering all the problems and algorithms without repetition [2]. We provide a brief description of each implementation's pros and cons below at the end of this section, however, further details can be found in the corresponding subsections.

7.1. Vanilla QAOA

Regardless of the problem addressed, the vanilla QAOA implementation requires a cost Hamiltonian, a mixer Hamiltonian, and a QAOA Circuit. These were all implemented following the instructions provided by the Qiskit Textbook (4). Here we focus on the Number Partitioning problem to illustrate the steps and the code.

7.1.1. Hamiltonians The hyperparameters γ, β are initialized to 0.5. These parameters will be adjusted during the optimization phase for the QAOA circuit. For an arbitrary QUBO in the form of

$$\sum_{i,j=1}^{n} x_i Q_{ij} x_j + \sum_{i=1}^{n} c_i x_i,$$

the cost Hamiltonian was created by implementing RZ gates across all angles θ_1:

$$\theta_1 = \frac{\gamma}{2}(\sum_{i=1}^{n} c_i + \sum_{i=1}^{n} Q_{ij}).$$

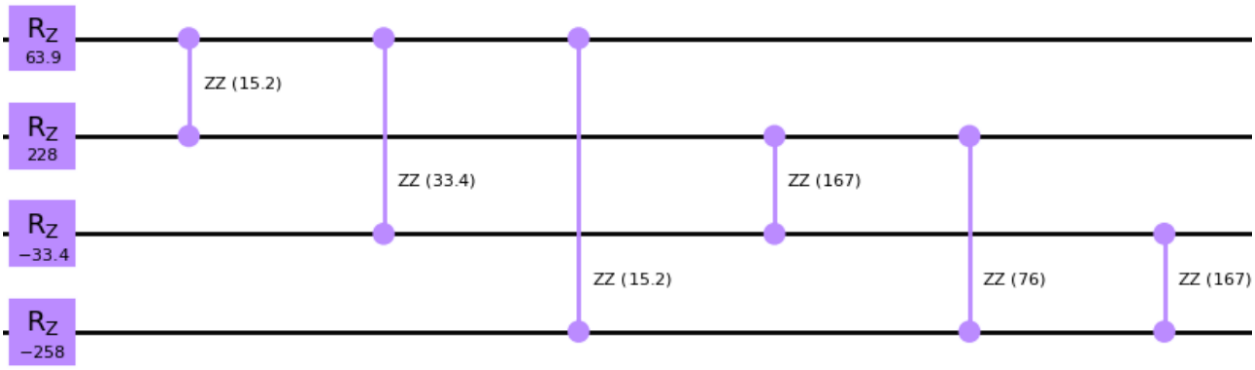
This layer of RZ gates is followed by a layer of ZZ gates applied on angles θ_2:

$$\theta_2 = \frac{1}{4} Q_{ij} \gamma.$$

The corresponding code and circuit gates are shown below:

```
def cost_H(gammas, quadratics, linears):
  qc = QuantumCircuit(num_qubits, num_qubits)
  for i in range(len(linears)):
    qc.rz(1/2*(linears[(0, i)]+sum(quadratics[(i, j)] for j in range(
       num_qubits)))*gammas, i)

  for (i, j) in quadratics.keys():
    if i!=j:
      qc.rzz((1/4)*quadratics[(i, j)]*gammas, i, j)

  qc.barrier()
  return qc
```

[2] The notebooks can in the following github: https://github.com/arulrhikm/Solving-QUBOs-on-Quantum-Computers.git

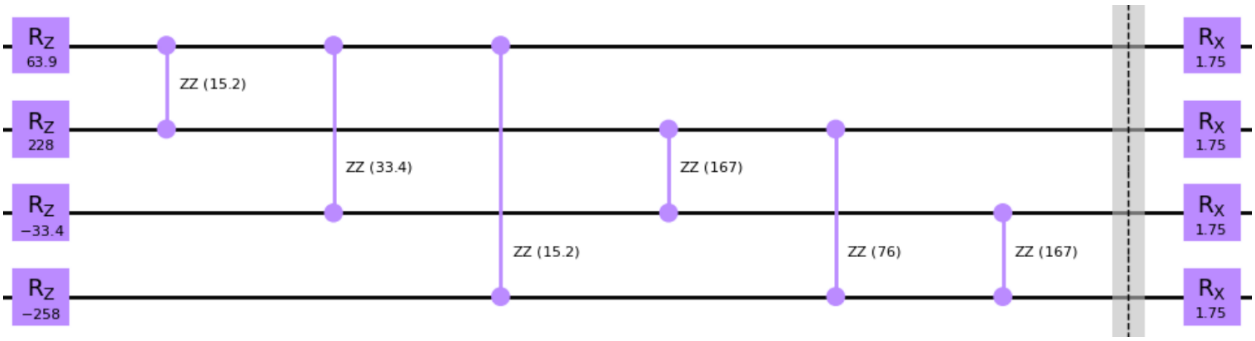


The mixer Hamiltonian was created by applying RX gates to angle θ_3:

$$\theta_3 = 2\beta.$$

The code and the resulting circuit (with both cost and mixer Hamiltonians) is shown below:

```
def mixer_H(betas):
    qc = QuantumCircuit(num_qubits, num_qubits)
    for i in range(num_qubits):
        qc.rx(2*betas, i)
    qc.barrier()
  return qc
```



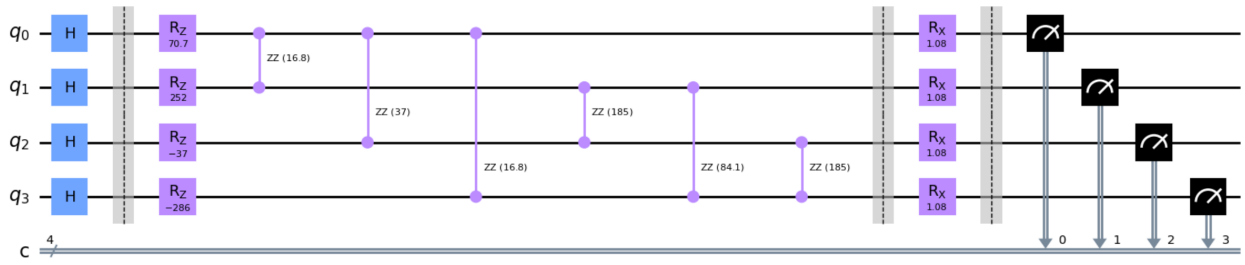
7.1.2. QAOA Circuit Once the generic framework is set up, the Number Partitioning QUBO quadratic and linear coefficients are substituted to replace the generic Cost and mixer Hamiltonians implementations. The initial model is created using DOcplex (10), then converted into a QUBO, and then partitioned into quadratic and linear terms to implement each gate. The test example has $\{1, 5, 5, 11\}$ as the set to be partitioned.

```
def circuit(gammas, betas, quadratics, linears):
  circuit = QuantumCircuit(num_qubits, num_qubits)
  assert(len(betas) == len(gammas))
```

```
  p = len(betas)

  for i in range(num_qubits):
    circuit.h(i)
  circuit.barrier()

  for i in range(p):
    circuit &= cost_H(gammas[i], quadratics, linears)
    circuit &= mixer_H(betas[i])

  circuit.measure(range(num_qubits), range(num_qubits))

  return circuit
```

The following code generates the number partitioning instance, converts it to the QUBO model and then collects the parameters of the QUBO model to construct the final circuit (with p = 1 for conciseness).

```
test = [1, 5, 11, 5]
arr = test
n = len(arr)
c = sum(arr)

model = Model()
x = model.binary_var_list(n)
H = (c - 2*sum(arr[i]*x[i] for i in range(n)))**2
model.minimize(H)
problem = from_docplex_mp(model)

converter = QuadraticProgramToQubo()
qubo = converter.convert(problem)

quadratics_coeffs = qubo.objective.quadratic.coefficients
linears_coeffs = qubo.objective.linear.coefficients
constant = qubo.objective.constant

num_qubits = qubo.get_num_vars()
```



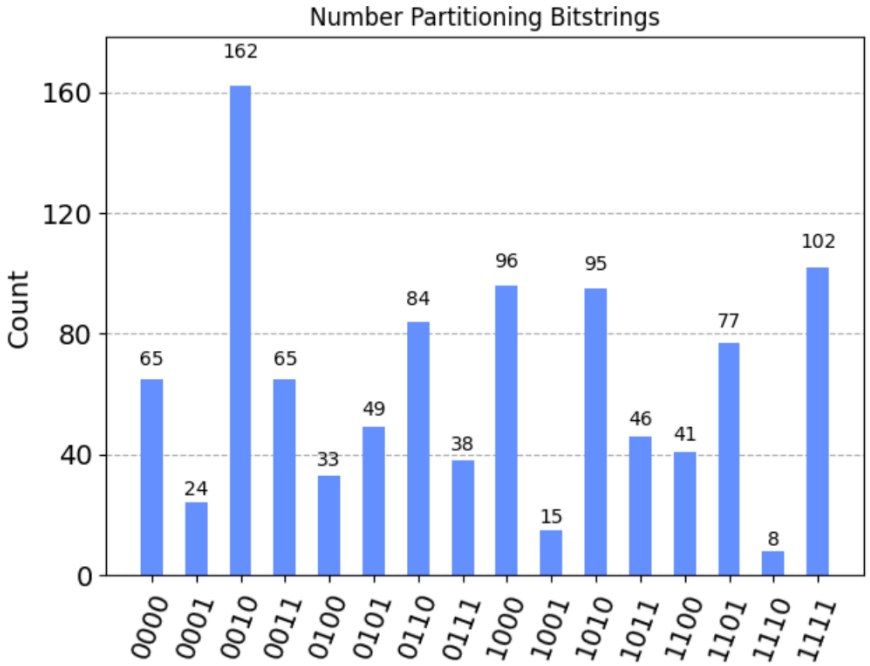
Once the circuit is created, its γ and β parameters are optimized using a classical optimizer. To optimize, we define an objective function and expectation function for the outputs of the circuit. The objective function

measures the quality of QAOA output solutions while the expectation function averages these solutions. The optimizer adjusts the parameters to minimize the expected value.

```
def npp_obj(str):
  sum_0 = 0
  sum_1 = 0
  for i in range(len(str)):
    if str[i] == '0':
      sum_0 += arr[i]
    else:
      sum_1 += arr[i]
  return abs(sum_0-sum_1)
```

```
def npp_expectation(thetas):
    backend = Aer.get_backend('qasm_simulator')
    gammas = theta[:int(len(thetas)/2)]
    betas = theta[int(len(thetas)/2):]
    pqc = circuit(gammas, betas, quadratics, linears)
    counts = execute(pqc, backend, shots=1000).result().get_counts()
    best_sol = max(counts, key=counts.get)

    return npp_obj(best_sol)
```

After the optimization process is completed, the optimal parameters are passed to the final QAOA circuit and sampled from 1000 shots. The number of shots can be adjusted, however, more shots will lead to more accurate results. The sampled outputs are displayed in the histogram below:



The resulting binary strings correspond to the partition of the original array. For a binary string output `s` and original array `arr`, if `s[i]` is 1, `arr[i]` is placed in set A; if `s[i]` $= 0$, it is into $S \setminus A$. As visible from the histogram, the most probable outcome is a perfect partition with $c = 22$ split into $A = \{11\}$ and $\{1, 5, 5\} = S \setminus A$.

7.2. OpenQAOA

OpenQAOA (36) is a multi-backend SDK used to easily implement QAOA circuits. It provides simple yet very detailed implementations of QAOA circuits that can not only run on IBMQ devices and simulators but also on Rigetti Cloud Services (20), Amazon Braket (3), and Microsoft Azure (19). The parent company of OpenQAOA also provides usage of their own custom simulators.

We illustrate the implementation for the *Max-Cut* Problem. This process requires the same steps as the vanilla QAOA implementation. However, most of the technical challenges are circumvented by the OpenQAOA API. Like with the vanilla implementation, the first step is to create a problem instance and then implement the QUBO model for it. The graph instance creation and the QUBO creation implementations are shown below:

```
def draw_graph(G, colors, pos):
    default_axes = plt.axes()
    nx.draw_networkx(G, node_color=colors, node_size=600, alpha=0.8, ax
       =default_axes, pos=pos)
    edge_labels = nx.get_edge_attributes(G, "weight")
```

```
G = nx.generators.fast_gnp_random_graph(n=6, p=0.5)
n = 6
colors = ["r" for node in G.nodes()]
pos = nx.spring_layout(G)
draw_graph(G, colors, pos)
```

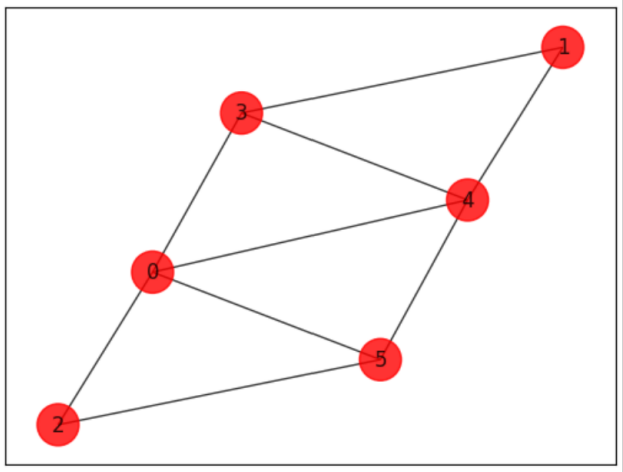


Figure 5 Generated Max-Cut Graph Instance

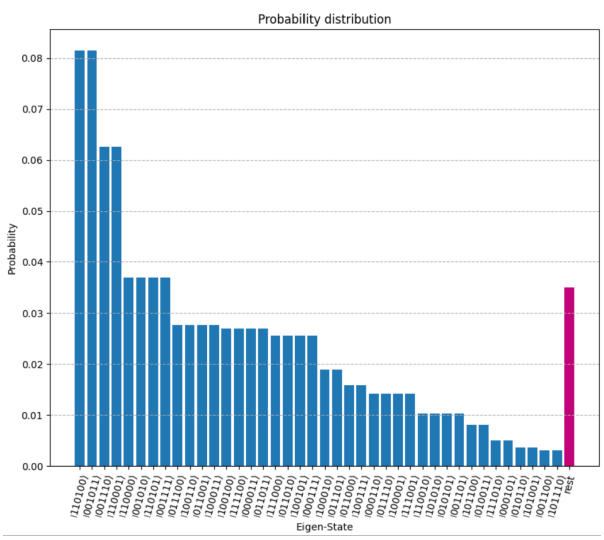
The QUBO creation process for Max-Cut is very similar to the process with Number Partitioning.

```
model = Model()
x = model.binary_var_list(n)
H = sum(2*x[e[0]]*x[e[1]] - x[e[0]] - x[e[1]] for e in G.edges)
model.minimize(H)

# Converting the Docplex model into its qubo representation
qubo = FromDocplex2IsingModel(model)

# Ising encoding of the QUBO problem
maxcut_ising = qubo.ising_model
```

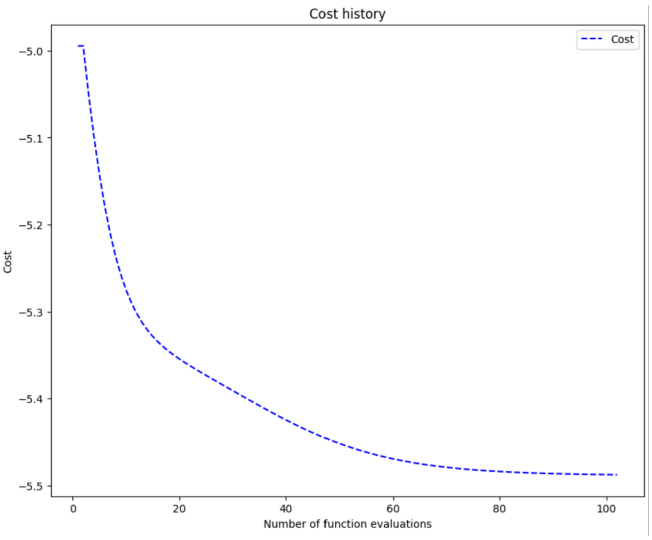
7.2.1. OpenQAOA Circuit Once the QUBO is created, OpenQAOA provides a short and easy implementation to create the QAOA circuit. To start, you just need to initialize the QAOA object (`q = QAOA()`) and then define its properties. These properties include whether the circuit will be run locally on a simulator or virtually on a simulator or real device, the configurations of the cost and mixer Hamiltonians, the number of repetitions of the hamiltonians, the classical optimizer and its properties, the number of shots to simulate the circuit, and much more. Once all the circuit characteristics are determined, the user can compile and optimize parameters for the circuit.

```
# initialize model
q = QAOA()

# device
q.set_device(create_device('local', 'vectorized'))

# circuit properties
q.set_circuit_properties(p=2, param_type='standard', init_type='rand',
   mixer_hamiltonian='x')

# backend properties
q.set_backend_properties(n_shots = 1000)

# set optimizer and properties
q.set_classical_optimizer(method='vgd', jac="finite_difference")

q.compile(maxcut_ising)

q.optimize()
```

7.2.2. Analysis Tools Once the circuit is created and executed, various visualization tools can be instantly implemented. These include a histogram of bitstring outcomes and their frequencies and the classical optimization plot. After the most probable bitstring output is post-processed, we obtain the optimal partition graph for the Maximum Cut.



(a) Probability Distribution of Bitstrings



(b) Classical Optimization Plot

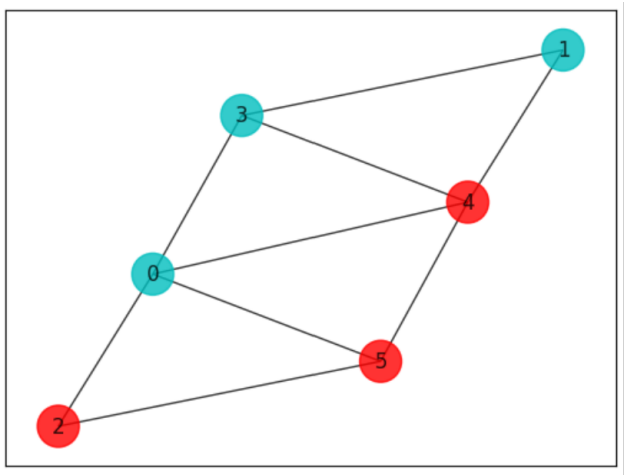


Figure 7 Optimal Graph Partitioning (Max Cut) based on Most Common Bitstring: (Nodes 2, 4 and 5) and Nodes (0, 1, 3)

7.3. QAOA via Qiskit

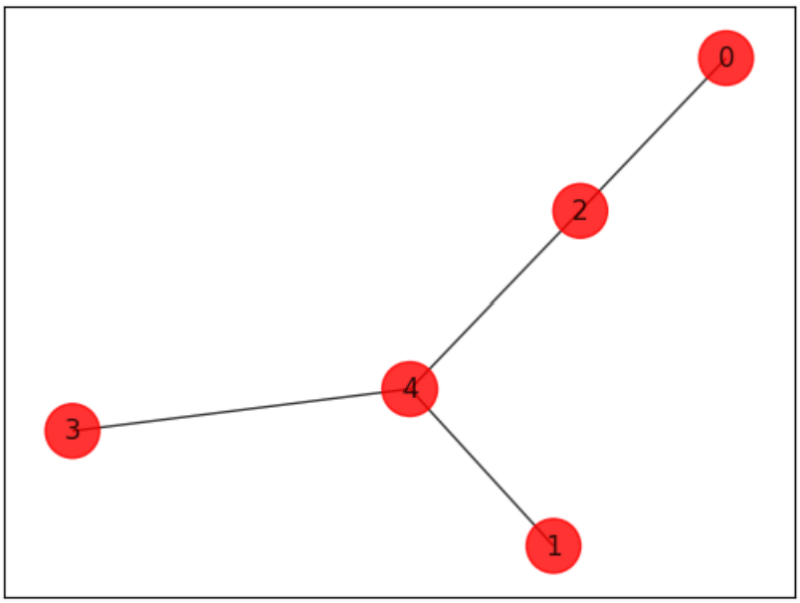
Qiskit-Optimization is a part of IBM Qiskit's open-source quantum computing framework. It provides easy high-level applications of quantum algorithms that can run on both classical and quantum simulators. It provides a wide array of algorithms and built-in application classes for many canonical problems.

Qiskit Optimization is an extremely useful and easy way to solve QUBOs with various quantum algorithms including Variational Quantum Eigensolver (31), Adaptive Grover (17), and QAOA. Furthermore, because of compatibility with IBMQ, algorithms can run on actual IBM quantum backends. In the examples below, all code is run by default on the Qiskit QASM simulator. Although base-level implementation is easier, customization and in-depth analysis are more difficult. Tasks like plotting optimization history

or bitstring distributions, although possible, are significantly more challenging. Qiskit-Optimization's key limitations are its extremely long optimization times and limited set of optimizers.

Let us now consider the *Minimum Vertex Cover* problem. Similar to the other problems, the first step is to create a problem instance and its corresponding QUBO model.

```
# Generating a graph of 4 nodes

n = 5  # Number of nodes in graph
G = nx.Graph()
G.add_nodes_from(np.arange(0, n, 1))
edges = [(0, 2, 1.0), (2, 4, 1.0), (1, 4, 1.0), (3, 4, 1.0)]
# tuple is (i,j,weight) where (i,j) is the edge
G.add_weighted_edges_from(edges)

colors = ["r" for node in G.nodes()]
pos = nx.spring_layout(G)

draw_graph(G, colors, pos)
```




```
model = Model()
x = model.binary_var_list(n)
P = 10
H = sum(x[i] for i in range(n)) + P*sum(1 - x[e[0]] - x[e[1]] + x[e
    [0]]*x[e[1]] for e in edges)
model.minimize(H)
problem = from_docplex_mp(model)
qubo = QuadraticProgramToQubo().convert(problem)
```

Similar to the Vanilla and OpenQAOA implementations, the following steps are to set up the QAOA circuit and optimize it. Because of the built-in capabilities of the Qiskit-Optimization, this requires very little extra effort. Similar to with OpenQAOA, we just need to define a QAOA object. This object takes in a sampler, optimizer, and number reptitions and its properties are harder to modify.

```
nelder_mead = NELDER_MEAD(maxiter=250)
sampler = Sampler()
qaoa = QAOA(sampler=sampler, optimizer=nelder_mead, reps=2)
algorithm = MinimumEigenOptimizer(qaoa)
```

Thus we obtain the following solution Minimum Vertex Cover graph.

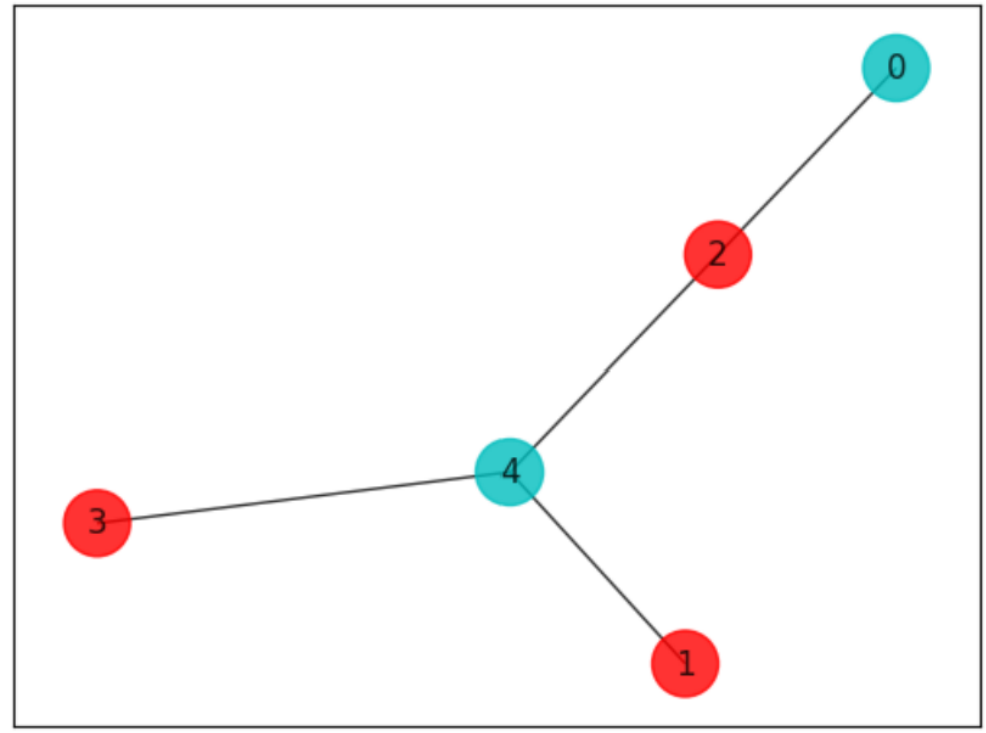


Figure 8 Optimal Minimum Vertex Cover: Nodes 0 and 4

7.4. Simulated Annealing

Let us now turn to we cover is Simulated Annealing which is used to solve the Order Partitioning Problem. Like the other algorithms, the first step is creating the problem instance and its corresponding QUBO model. As the Order Partitioning problem is new, there are no pre-existing datasets or problem instances to use. Thus, we create a problem instance as shown below:

```
Stocks = ['A', 'B', 'C', 'D', 'E', 'F']
stock_vals = [300, 100, 100, 200, 200, 100]
risk_factor_matrix = [[0.3, 0.1, 0.1, 0.2, 0.2, 0.1],
                      [0.4, 0.05, 0.05, 0.12, 0.08, 0.3],
```

```
                        [0.1, 0.2, 0.2, 0.3, 0.05, 0.05]]
T = sum(stock_vals)
n = 6 # number of stocks
m = 3 # number of risk factors
```

```
x = Array.create('x', n, 'BINARY')
H1 = (T - 2*sum(stock_vals[j]*x[j] for j in range(n)))**2
H2 = sum(sum(risk_factor_matrix[i][j]*(2*x[j]-1)**2 for j in range(n))
    for i in range(m))

# Construct hamiltonian
a = Placeholder("a")
b = Placeholder("b")
H = a*H1 + b*H2
model = H.compile()

# Generate QUBO
feed_dict = {'a': 2, 'b': 2}
bqm = model.to_bqm(feed_dict=feed_dict)
```

Once the Order Partitioning BQM is created, it is then run on the Simulated Annealing Sampler. Additional parameters can be modified in the Simulated Annealing Sampler such as the beta range, beta schedule, number of sweeps per beta, etc. where beta is the inverse temperature. For simplicity, we use the default parameters and read from the sampler 10 times. Less reads are required than with Quantum Annealing since the algorithm converges to optimal solutions more efficiently. Just like with Quantum Annealing, the solutions with the lowest energy correspond to optimal outcomes.

```
# Getting Results from Sampler
sa = neal.SimulatedAnnealingSampler()
sampleset = sa.sample(bqm, num_reads = 10)
decoded_samples = model.decode_sampleset(sampleset, feed_dict=feed_dict
    )
sample = min(decoded_samples, key=lambda x: x.energy)
```

Similar to quantum annealing, the binary solution output from simulated annealing still needs to be sorted and paired with the corresponding stock. This is done explicitly in the associated Github. After this post-processing, we obtain the following solution:

```
Stock Partition:  ['A', 'B', 'C'] ['D', 'E', 'F']
Difference of Net Cost between Partition:  0
Difference of Net Risk between Partition:  0.1
```

Figure 9 Solution to Order Partitioning instance

7.5. Quantum Annealing

The final algorithm discussed is Quantum Annealing. This algorithm was implemented using D-Wave's Quantum Annealer and is applied to the Cancer Genomics application. Due to the higher number of qubits available here as compared to IBM devices, larger instances can be executed.

The Cancer Genomics problem requires significant preprocessing effort. First, to construct the cancer pathway QUBO, the disease data must be read from some external source. For this tutorial, the data was collected from The Cancer Genome Atlas Acute Myeloid Leukemia dataset posted on cBioPortal (24).

```
# import necessary package for data imports
from bravado.client import SwaggerClient
from itertools import combinations

# connects to cbioportal to access data
cbioportal = SwaggerClient.from_url('https://www.cbioportal.org/api/v2/
    api-docs', config={"validate_requests":False,"validate_responses":
    False,"validate_swagger_spec":False})

# accesses cbioportal's AML study data
aml = cbioportal.Cancer_Types.getCancerTypeUsingGET(cancerTypeId='aml')
    .result()

# access the patient data of AML study
patients = cbioportal.Patients.getAllPatientsInStudyUsingGET(studyId='
    laml_tcga').result()

# for each mutation, creates a list of properties associated with the
    mutation include geneID, patientID, and more
InitialMutations = cbioportal.Mutations.
    getMutationsInMolecularProfileBySampleListIdUsingGET(
     molecularProfileId='laml_tcga_mutations',
     sampleListId='laml_tcga_all',
     projection='DETAILED'
).result()
```

Using this data, a Patient-Gene dictionary is constructed, which contains the cancerous genes that each patient has. The first four entries of the dictionary is shown below:

```
Patient-Gene Dictionary:
TCGA-AB-2802
['IDH1', 'PTPN11', 'NPM1', 'MT-ND5', 'DNMT3A']
TCGA-AB-2804
['PHF6']
TCGA-AB-2805
['IDH2', 'RUNX1']
TCGA-AB-2806
['KDM6A', 'PLCE1', 'CROCC']
```

Figure 10 Patient-Gene Dictionary sample

Note that with this dictionary both the patients and genes are enumerated. Thus the dictionary is used to create a diagonal matrix $\mathbf{D}$ where index d_{ii} of $\mathbf{D}$ represents the number of instances of gene i amongst the all the patients.

```
# creating diagonal matrix D
import numpy as np
D = np.zeros((n, n))
for i in range(n):
    count = 0
    for k in PatientGeneDict.keys():
        if geneList[i] in PatientGeneDict[k]:
            count += 1
    D[i][i] = count
```

To create the weighted adjacency matrix $\mathbf{A}_w$, the Patient-Gene dictionary is modified to contain Gene Pairs. For example, if a patient possesses gene i and gene j, they possess the gene pair (i, j). We first create a helper function to create all pairs of entries of a list and then use a dictionary to map each patient to its list of gene pairs. This new Patient-Gene Pair dictionary is then used to create matrix $\mathbf{A}$ where index A_{ij} represents the number of patients who possess gene pair (i, j). Note by symmetry, this quantity is also stored at A_{ji}.

```
# helper function to generate all gene pairs from a list of genes
def generate_pairs(list):
    pairs = set()
    for subset in combinations(list, 2):
        pairs.add(tuple(sorted(subset)))
    return pairs
```

```
# creates a patient-(gene-list-pair) dictionary
PatientGeneDictPairs = {}
for m in mutations:
        PatientGeneDictPairs[m.patientId] = generate_pairs(
           PatientGeneDict[m.patientId])
```

```
# creates the A exclusivity matrix
A = np.zeros((n, n))
for j in range(n):
    if i != j:
        count = 0
        for k in PatientGeneDict.keys():
            if (geneList[i], geneList[j]) in PatientGeneDictPairs[k]:
                count += 1
        A[i][j] = count
        A[j][i] = count
```

Once the **D** and **A** matrices are constructed, we create the QUBO model and later the Binary Quadratic Model (BQM) to run on D-Wave. This BQM is unconstrained as it uses unconstrained binary variables to model the pathway problem.

```
# initializes an array of QUBO variables
x = Array.create('x', n, 'BINARY')
H1 = sum(sum(A[i][j]*x[i]*x[j] for j in range(n)) for i in range(n))
H2 = sum(D[i][i]*x[i] for i in range(n))
a = Placeholder("alpha")
H = H1 - a*H2

# creates a Hamiltonian to run on D-Wave sampler
model = H.compile()
feed_dict = {'alpha': 0.45}
bqm = model.to_bqm(feed_dict=feed_dict)
```

Once the BQM is created, it is then mapped to the qubits on the Quantum Processing Unit (QPU). The sampler does this process by encoding the problem to physical qubits and couplers. They then orchestrate the quantum annealing process to find the lowest energy solution. Recall the original form of a QUBO model:

$$\sum_{i<j} Q_{ij}x_i x_j + \sum_i Q_{ii}x_i.$$

Embedding is the process of mapping the problem to a graph where Q_{ii} is the bias of the nodes (the physical qubits) and Q_{ij} is the weight of the edges (the couplers). This mapping from the binary quadratic model to the QPU graph is known as minor embedding however other types of embeddings do exist for different problems. Composites allow higher-level customization of the annealing process allowing options like parallel QPU processing, automatic embedding, etc. The Ocean API is very powerful and provides a plethora of customizable features for samplers, embeddings, and composites, but for the sake of simplicity, we just use the standard D-Wave Sampler and Embedding Composite.

```
# Getting Results from Sampler
sampler = EmbeddingComposite(D-WaveSampler())
sampleset = sampler.sample(bqm, num_reads=1000)
sample = sampleset.first
```

The annealing process will return for each read, so we call "sample.first" to return the one with the lowest energy. Typically the variables are unordered, so they must be sorted to correspond correctly to the genes. This post-processing is tedious and included in the associated Github, but if implemented correctly should provide a Cancer Genome Pathway (one example shown below). As discussed in the Cancer Genomics section before, we also provide the of properties of this Cancer Genome Pathway (coverage, coverage/gene, independence, measure).

```
['ASXL1', 'BRINP3', 'DNMT3A']
coverage: 61.0
coverage/gene: 20.33
indep: 4.0
measure: 5.08
```

Figure 11 Cancer Gene Pathway discover through Quantum Annealing

Implementation	Characteristics
QAOA (Vanilla)	• High Circuit Customization • High Optimizer Customization • Hard Implementation • Accuracy substantially decreases with problem size
QAOA (OpenQAOA)	• Moderate Circuit Customization • Moderate Optimizer Customization • Easy Implementation • Best Time vs Accuracy tradeoffs of QAOA implementations
QAOA (Qiskit)	• Low Circuit Customization • Low Optimizer Customization • Easy Implementation • Time exponentially increases with problem size
Simulated Annealing	• High Parameter Customization • Handle Larger Problem Size • Fastest Runtime • Most Accurate Results • Easy Implementation
Quantum Annealing	• High Topology Customization (not discussed in this paper) • Handle Larger Problem Size • Fastest Quantum Runtime • Most Accurate Quantum Results • Easy Implementation

8. Concluding Remarks

The purpose of this tutorial was to rapidly introduce to a novice how to (a) represent canonical and practical problems as Quadratic Unconstrained Binary Optimization (QUBO) models and (b) solve them using (i) Gate/Circuit quantum computers (such as IBM) using Quantum Approximate Optimization Algorithm (QAOA), (ii) Quantum Annealing (on D-Wave) and (iii) Simulated Annealing (classical). We illustrated the above algorithms on three canonical problems - Number Partitioning, Max-Cut, and Minimum Vertex Cover

and two practical problems (one from cancer genomics and the other from a Hedge Fund portfolio management) (38). To take better advantage of these nascent devices, it is necessary to develop decomposition methods, an active area of research (40), and a topic for a future tutorial.

Acknowledgements. The authors thank Claudio Gomes, Anil Prabhakar, Elias Towe, Shrinath Viswanathan, and Daniel Mai for their comments on the previous version.

References

[1] Hedayat Alghassi, Raouf Dridi, A. Gordon Robertson, and Sridhar Tayur. Quantum and quantum-inspired methods for de novo discovery of altered cancer pathways. *bioRxiv*, 2019.

[2] Bahram Alidaee, Fred Glover, Gary Kochenberger, and Cesar Rego. A new modeling and solution approach for the number partitioning problem. *JAMDS*, 9:113–121, 01 2005.

[3] Amazon Web Services. Amazon Braket, 2020.

[4] Various authors. *Qiskit Textbook*. Github, 2023.

[5] David E. Bernal, Sridhar Tayur, and Davide Venturelli. Quantum Integer Programming (QuIP) 47-779: Lecture Notes, 2021.

[6] Zhengbing Bian, Fabian Chudak, Robert Brian Israel, Brad Lackey, William G. Macready, and Aidan Roy. Mapping constrained optimization problems to quantum annealing with application to fault diagnosis. *Frontiers in ICT*, 3, 2016.

[7] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for Vertex Cover. In *Mathematical Foundations of Computer Science 2006*, pages 238–249, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] Harry Cohn and Mark Fielding. Simulated annealing: Searching for an optimal temperature schedule. *SIAM J. on Optimization*, 9(3):779–802, mar 1999.

[9] Clayton W. Commander. *Maximum cut problem, MAX-CUTMaximum Cut Problem, MAX-CUT*, pages 1991–1999. Springer US, Boston, MA, 2009.

[10] IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

[11] Diego de Falco and Dario Tamascelli. An introduction to quantum annealing. *RAIRO - Theoretical Informatics and Applications*, 45, 07 2011.

[12] Diego de Falco and Dario Tamascelli. An introduction to quantum annealing. *RAIRO - Theoretical Informatics and Applications*, 45(1):99–116, jan 2011.

[13] Krzysztof Domino, Mátyás Koniorczyk, Krzysztof Krawiec, Konrad Jałowiecki, Sebastian Deffner, and Bartłomiej Gardas. Quantum annealing in the nisq era: Railway conflict management. *Entropy*, 25(2), 2023.

[14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.

[15] Werry Febrianti, Kuntjoro Adji Sidarto, and Novriana Sumarti. Solving constrained mean-variance portfolio optimization problems using spiral optimization algorithm. *International Journal of Financial Studies*, 11(1), 2023.

[16] Bernd Gärtner and Jiří Matoušek. *Semidefinite Programming*, pages 15–25. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[17] Austin Gilliam, Stefan Woerner, and Constantin Gonciulea. Grover adaptive search for constrained polynomial binary optimization. *Quantum*, 5:428, April 2021.

[18] Fred Glover, Jin-Kao Hao, Gary Kochenberger, Mark Lewis, Zhipeng Lu, Haibo Wang, Yang Wang, Haibo Wang, and Yang Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of Combinatorial Optimization*, 28:58 – 81, 2014.

[19] Johnny Hooyberghs. *Azure Quantum*, pages 307–339. Apress, Berkeley, CA, 2022.

[20] Peter J Karalekas, Nikolas A Tezak, Eric C Peterson, Colm A Ryan, Marcus P da Silva, and Robert S Smith. A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quantum Science and Technology*, 5(2):024003, April 2020.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[22] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[23] Mark Leiserson, Dima Blokh, Roded Sharan, and Ben Raphael. Simultaneous identification of multiple driver pathways in cancer. *PLoS computational biology*, 9:e1003054, 05 2013.

[24] Timothy J. Ley, Christopher B. Miller, Li xia Ding, Benjamin J. Raphael, Andrew J. Mungall, A. Gordon Robertson, Katherine A. Hoadley, Timothy J. Triche, Peter W. Laird, J. David Baty, Lucinda L. Fulton, Robert S. Fulton, Sharon E. Heath, Joelle M. Kalicki-Veizer, Cyriac Kandoth, Jeffery M. Klco, Daniel C. Koboldt, Krishna L. Kanchi, Shashikant Kulkarni, Tamara L. Lamprecht, David E. Larson, Ling Lin, Charles Lu, Michael D. McLellan, Joshua F. McMichael, Jacqueline E. Payton, Heather K. Schmidt, David H. Spencer, Michael H. Tomasson, John W. Wallis, Lukas D Wartman, Mark A. Watson, John S S Welch, Michael C. Wendl, Adrian Ally, Miruna Balasundaram, Inanç Birol, Yaron S. N. Butterfield, Readman Chiu, Andy Chu, Eric Chuah, Hye-Jung E. Chun, Richard Corbett, Noreen Dhalla, Ranabir Guin, Ann He, Carrie Hirst, Martin Hirst, Robert A. Holt, Steven P. Jones, Aly Karsan, Darlene Lee, Haiyan Irene Li, Marco A. Marra, Michael Mayo, Richard A. Moore, Karen L. Mungall, Jeremy D.K. Parker, Erin D. Pleasance, Patrick Plettner, Jacquie E. Schein, Dominik Stoll, Lucas Swanson, Angela Tam, Nina Thiessen, Richard Varhol, Natasja H. Wye, Yongjun Zhao, S. Gabriel, Gad Getz, Carrie Sougnez, Lihua Zou, Mark D. M. Leiserson, Fabio Vandin, Hsin-Ta Wu, Frederick R. Applebaum, Stephen B. Baylin, Rehan Akbani, Bradley M. Broom, Ken Chen, Thomas C. Motter, Khanh Nguyen, John N. Weinstein, Nianziang Zhang, Martin L. Ferguson, Christopher Adams, Aaron D. Black, Jay Bowen, Julie M. Gastier-Foster, Tom Grossman, Tara M. Lichtenberg, Lisa Wise, Tanja Davidsen, John A. Demchok, Kenna R. Mills Shaw, Margi Sheth, Heidi J. Sofia, Liming Yang, James R. Downing, and Greg D. Eley. Genomic and epigenomic

landscapes of adult de novo acute myeloid leukemia. *The New England journal of medicine*, 368 22:2059–74, 2013.

[25] Zi Xuan Loke, Say Leng Goh, Graham Kendall, Salwani Abdullah, and Nasser R. Sabar. Portfolio optimization problem: A taxonomic review of solution methodologies. *IEEE Access*, 11:33100–33120, 2023.

[26] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2, 2014.

[27] Catherine C McGeoch. *Adiabatic Quantum Computation and Quantum Annealing*. Morgan&Claypool, Kenfield, CA, 2014.

[28] Stephan Mertens. Phase transition in the number partitioning problem. *Phys. Rev. Lett.*, 81:4281–4284, Nov 1998.

[29] Stephan Mertens. The easiest hard problem: Number partitioning, 2003.

[30] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 2011.

[31] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), July 2014.

[32] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[33] Atanu Rajak, Sei Suzuki, Amit Dutta, and Bikas K. Chakrabarti. Quantum annealing: an overview. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 381(2241), dec 2022.

[34] Eleanor Rieffel and Wolfgang Polak. *Quantum Computing: A Gentle Introduction*. The MIT Press, Cambridge, MA, 2014.

[35] R.A. Rutenbar. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26, 1989.

[36] Vishal Sharma, Nur Shahidee Bin Saharan, Shao-Hen Chiew, Ezequiel Ignacio Rodríguez Chiacchio, Leonardo Disilvestro, Tommaso Federico Demarie, and Ewan Munro. Openqaoa – an sdk for qaoa, 2022.

[37] M. SHIFMAN. QUANTUM TUNNELING. In *Multiple Facets of Quantization and Supersymmetry*, pages 52–67. WORLD SCIENTIFIC, oct 2002.

[38] ManMohan S. Sodhi and Sridhar R. Tayur. Make Your Business Quantum-Ready Today. *Management and Business Review*, 2023.

[39] Darren Strash. On the power of simple reductions for the maximum independent set problem. In Thang N. Dinh and My T. Thai, editors, *Computing and Combinatorics*, pages 345–356, Cham, 2016. Springer International Publishing.

[40] Sridhar Tayur and Ananth Tenneti. Quantum Annealing Research at CMU: Algorithms, Hardware, Applications. *Frontiers of Computer Science*, 2024.

[41] Simone Tibaldi, Davide Vodola, Edoardo Tignone, and Elisa Ercolessi. Bayesian optimization for qaoa, 2022.

[42] Fabio Vandin, Eli Upfal, and Benjamin J. Raphael. De novo discovery of mutated driver pathways in cancer. In Vineet Bafna and S. Cenk Sahinalp, editors, *Research in Computational Molecular Biology*, pages 499–500, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[43] Luzhi Wang, Shuli Hu, Mingyang Li, and Junping Zhou. An exact algorithm for minimum vertex cover problem. *Mathematics*, 7(7), 2019.

[44] Colin P. Williams. *Solving NP-Complete Problems with a Quantum Computer*, pages 293–318. Springer London, London, 2011.

[45] Junfei Zhao, Shihua Zhang, Ling-Yun Wu, and Xiang-Sun Zhang. Efficient methods for identifying mutated driver pathways in cancer. *Bioinformatics*, 28 22:2940–7, 2012.

[46] Qiang Zhou, Xiaojun Xie, Huan Dai, and Weizhi Meng. A novel rough set-based approach for minimum vertex cover of hypergraphs. *Neural Computing and Applications*, 34:21793 – 21808, 2022.