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

2025 INFORMS TUTORIALS
Vol. 00, No. 0, Xxxx 0000, pp. 000–000
ISSN 2694-4022, EISSN 2694-4030

Submitted to *2025 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 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. However, due to the nascency of the hardware, many of the envisioned speedups remain largely theoretical. Despite limitations of available quantum hardware, there is significant interest in exploring them. Although several articles and books (see Nielsen and Chuang (2011), 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 aims to provide quick access to IBM and D-Wave machines, thus covering two types of quantum algorithms, one based on the gate / circuit model (Rieffel and Polak 2014) and the other on quantum annealing (McGeoch 2014), 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). We then show how QUBOs model three canonical combinatorial optimization problems: Number Partitioning (Mertens 1998), Max-Cut (Commander 2009), and Minimum Vertex Cover (Chen et al. 2006). We next model two practical problems: Order Partitioning (for A/B testing at a hedge fund) and de novo discovery of driver genes in the

field of cancer genomics (Alghassi et al. 2019). We then provide step-by-step instructions [1] to solve QUBOs in two frameworks: gate / circuit model (IBM hardware (Qiskit contributors 2023)) and quantum annealing (D-Wave hardware (Bian et al. 2016)).

2. Mathematical Formulations for Optimization

This tutorial explores two key formulations for combinatorial optimization problems: Quadratic Unconstrained Binary Optimization (QUBO) and Ising Spin Glasses (Ising model). While the primary focus is on solving problems modeled as QUBOs, we will also provide a brief introduction to Ising models. Understanding the Ising model, the physics-based counterpart to QUBOs, is essential for appreciating their relevance in quantum computing. This connection highlights how problems in optimization can be directly mapped onto physical systems, enabling solutions through quantum techniques like quantum annealing.

2.1. Quadratic Unconstrained Binary Optimization (QUBO)

The Quadratic Unconstrained Binary Optimization (QUBO) model is a widely used model for optimization problems across various fields such as operations research, economics, finance, physics, and machine learning. The mathematical matrix form of a QUBO is:

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

where $\mathbf{x} \in \{0,1\}^n$ represents the solution vector, $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is a symmetric matrix and $c \in \mathbb{R}^n$ is a constant offset. (Note that c does not make any difference to the optimal solution.) After multiplying the matrix form, the resulting equation is a quadratic unconstrained binary optimization model (QUBO):

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

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

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

QUBO models were first introduced and systematically studied in the 1960s (Hammer and Rudeanu 1968), (Ginsburgh V. 1969). These models were initially popular due to their ability to represent various integer and combinatorial optimization problems, where some decision variables are subject to integer constraints. These combinatorial problems include NP-Hard problems which, by definition, have no known polynomial-time solving algorithm, so QUBO models provide researchers with an alternative equivalent way to study these types of problem.

Historically, a variety of classical algorithms have been employed to solve QUBO problems. These include branch-and-bound (Pardalos and Rodgers 1990), semidefinite optimization (Helmberg and Rendl

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

1998), and interior point methods for Q matrices that are positive semidefinite (Abello et al. 2001). More relevant to this work is that many heuristic and meta-heuristic algorithms such as simulated annealing (Kirkpatrick et al. 1983a), Genetic Algorithm (Gerges et al. 2018), and Tabu Search (Glover 1989) perform well in finding high-quality solutions for QUBO problems. Furthermore, despite these algorithms being designed for general QUBO problems and not particular combinatorial optimization problems, these solvers often outperform the specialized methods in terms of solution quality and runtime.

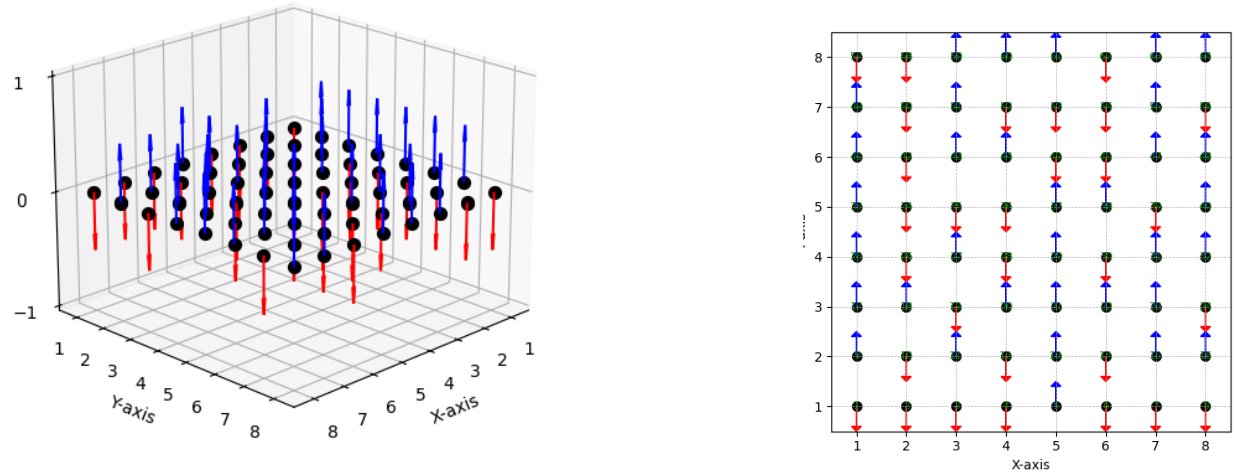
In recent years, the rise of quantum computing has sparked significant interest in QUBO models, driven by advancements in both gate-based (Farhi et al. 2014) and annealing-based (Kadowaki and Nishimori 1998) quantum heuristics. More specifically, QUBOs have drawn attention due to their mathematical equivalence to physics-based Ising models.

2.2. Ising Models

Ising models are mathematical tools originally introduced by Ernst Ising and Wilhelm Lenz to model ferromagnetism in statistical physics. The Ising model is defined by the Hamiltonian:

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

The system modeled by the Ising formulation is typically represented as a lattice graph. Each spin variable $\sigma_i \in \{-1,1\}$ corresponds to the state of the node i in the graph. The coefficients J_{ij} represent the strength of the interaction between nodes i and j, while h_i denotes the external bias applied to node i. The energy function $H(\sigma)$ evaluates the total energy of a configuration (modeled with $\sigma = (\sigma_1, \sigma_2 \ldots \sigma_n)$) by summing over the two-body interaction terms $J_{ij}\sigma_i\sigma_j$ and the bias contributions $h_i\sigma_i$.



(a) Ising Model of a Ferromagnet (b) Top view of Ising Model

Figure 1 Two-Dimensional Square-Lattice Ising model

The Ising model has is directly applied in quantum computing, particularly in the context of quantum annealing. In D-Wave quantum computers, the nodes represent qubits with their spin σ_i and the physical realization of this model is achieved through couplers that encode the interaction coefficients J_{ij} by entangling qubits [2] and through magnetic biases expressed through the h_i terms. Together, the biases and couplers define an energy landscape in which D-Wave quantum computers find the minimum energy during quantum annealing.

Ising models and QUBO models can easily be mapped to each other through bijective mapping of QUBO binary variables to Ising spin variables. QUBO binary variables can be mapped to Ising spin variables using:

$$\sigma_i = 2x_i - 1$$

and Ising spin variables can be mapped to QUBO binary variables using:

$$x_i = \frac{\sigma_i + 1}{2}$$

The simple physical implementation of Ising models and the direct correspondence between the Ising model and QUBO (Quadratic Unconstrained Binary Optimization) motivates the use of QUBO formulations for framing optimization problems suitable for quantum computing platforms. The

These quantum algorithms tackle problems using fundamentally different paradigms compared to classical methods, offering potential for both analytical insights and computational speedups. Although quantum heuristics are unlikely to provide exponential speed-ups to NP-Hard problems (Preskill 2018), they have demonstrated promise in delivering quadratic speed-ups and improved heuristic approaches for certain domains of problems (Guerreschi and Matsuura 2019).

3. Three Canonical Problems

We have chosen three well-known canonical problems to formulate as QUBO models and solve with quantum algorithms. These three 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. Readers can refer to literature that describes different techniques to benchmark with the approaches in this paper (Lucas 2014) , (Karp 1972), (Kikuura et al. 2024). There are no known exact classical polynomial-time solutions to these problems, making them viable 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:

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

[2] The definition of entanglement is further explain in section 5.1.6

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.

This objective function can be expressed as a QUBO using the following binary variable: $x_i = 1$ indicates that s_i belongs to A and if $x_i = 0$ then 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 in 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:

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

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 studied in computer science (Alidaee et al. 2005), (Mertens 2003).

3.2. Max Cut

The Max-Cut problem is the following: Given an undirected graph, G with a vertex set V and an 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 7 for a graph with six nodes labeled $0, 1, 2, 3, 4, 5$ and eight edges.

Similarly to Number Partitioning, the QUBO can be constructed by setting $x_i = 1$ if the vertex i is in A and letting $x_i = 0$ if vertex i is in $V \setminus A$. The expression $x_i + x_j - 2x_i x_j$ identifies whether the edge $(i, j) \in E$ is in the cut: $x_i + x_j - 2x_i x_j$ 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:

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

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

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

Note that this can be expressed as

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

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 of the quintessential combinatorial optimization problems (Commander 2009) that has been studied in a variety of contexts, notably being the target of different approximation algorithms such as the famous Goemans-Williamson algorithm (Goemans and Williamson 1995) that inspired the semidefinite programming paradigm (Gärtner and Matoušek 2012) and the original QAOA algorithm (Farhi et al. 2014).

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 property can be expressed as the term

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

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 term and the use of a penalty factor P:

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

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

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

Unlike the other formulations, we have a penalty factor P that determines how strictly the covering criteria must be enforced. A higher P will ensure that 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 (Wang et al. 2019), (Zhou et al. 2022).

4. Two Practical Problems

We now introduce two practical problems. The Ordering Partitioning problem is described for the first time in this paper. It is different, but related to the well-studied Portfolio Optimization problem (Loke et al. 2023), (Febrianti et al. 2023), and is a natural extension of the Number Partitioning problem, so it offers a graceful transition from canonical problems to practical ones. The Cancer Genomics problem identifies altered cancer pathways using mutation data from the Cancer Genome Atlas (TCGA) acute myeloid leukemia (AML) data set and is related to a different canonical problem, the Maximum Independent Set problem, which reduces to the Vertex Cover problem (Alghassi et al. 2019) (Ley et al. 2013).

4.1. Order Partitioning

Order[3] Partitioning is used for A/B testing of various investment strategies (proposed by hedge fund researchers). 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 to Number Partitioning and thus is an NP-complete. This class of problems by definition has no good polynomial-time algorithms and is a candidate for quantum speedup (Williams 2011).

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 the 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 would first like to reduce the absolute risk on each stock before minimizing its overall stocks. Considering binary spin variables σ_i that only take values in $\{-1, 1\}$, the minimum absolute risk is 0. This case is similar to the Number Partitioning problem where 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 in all stocks, we get the following final expression to minimize across risk factors in terms of the spin variables of Ising:

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

Converting back to QUBO variables and weighting with the 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.

[3] We retain the industry term 'order' for a block of stock that is traded. In our context of A/B testing, these blocks cannot be split.

4.2. Cancer Genomics using TCGA

The second application is the 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 cancer. Its identification is useful for understanding the disease and developing treatments. This problem nontrivially reduces to the Independent Set (which is well known to reduce from the Vertex Cover (Strash 2016)). Thus, this practical problem is NP-complete and is a suitable candidate for quantum speedup.

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 representing the hyperedge. Note that this incidence matrix is of the form

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

where 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:

$$L^+ = BB^{\mathrm{T}}$$

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

- Coverage: We want to identify which genes are the 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 on the patient's gene list, it is less likely there will be another. This is not a hard rule and is sometimes violated.

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

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

- Degree Matrix $\mathbf{D}$: This diagonal matrix corresponds to the coverage criterion. Each index d_{ij}, where $i = j$ represents the number of patients affected by the 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 by trying to find just one pathway. We define a solution pathway as $\mathbf{x} = \begin{pmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{pmatrix}^{\mathrm{T}}$ where for all $i \in \{1, 2, 3, \dots, n\}$, x_i is a binary variable. If $x_i = 0$, then the 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 $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$ and the coverage term

is $\mathbf{x}^{\mathrm{T}}\mathbf{D}\mathbf{x}$. 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

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

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 values a_{ij} and d_{ij} of A and D, respectively, as follows:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j - \alpha\sum_{i=1}^{n} d_i x_i.$$

This QUBO can be extended to identify multiple cancer pathways in a single run (Leiserson et al. 2013). Note that this is just one possible formulation of formulating the cancer pathway problem however other approaches do exist. See for example (Vandin et al. 2011), (Zhao et al. 2012).

5. Quantum Computing: A Primer

We provide a very succinct introduction to quantum computing. There are two main paradigms for 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 a core algorithm that we will thoroughly investigate and implement. The gate-based model relies on the QAOA 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. Gate-based Quantum Computing

Gate-based Quantum Computing is the most widely used and researched version of quantum computing. It has clear analogs to digital computing (qubits $\leftrightarrow$ bits, quantum gates $\leftrightarrow$ logic gates) and is a general-purpose universal quantum computing paradigm that uses discrete sequences of quantum gates. Current hardware implementations use superconducting qubits and trapped ions systems.

5.1.1. Qubits Similarly 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 underlying mathematics goes beyond the scope of the paper, however, a Hilbert Space can be understood as a special type of (possibly infinite dimensional) vector space[4].

[4] See an earlier tutorial by Siddhu and Tayur (2022) for details.

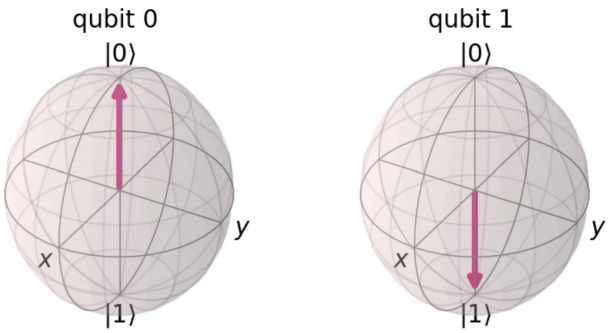


Figure 2 Bloch Sphere representations of $|0\rangle$ and $|1\rangle$ respectively

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": $|\psi\rangle$. Their adjoints are known as "bras": $\langle\psi| = \overline{|\psi\rangle}^{\mathrm{T}}$ and are also frequently used in calculations. This notation is known as the Dirac notation and is commonly used in quantum mechanics.

The coefficients α and β are complex-valued probability amplitudes that, when squared, represent the probability of measuring each state. This concept is known as Born's Rule and can be used to determine the probability of measuring each basis state:

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

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

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 express n states. This ability for qubits to map multiple states is known as quantum superposition and contributes to quantum speedup.

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

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

5.1.2. Quantum Gates Qubits are manipulated using unitary operators U. These operators are defined by two key properties:

1. They only map unit vectors to unit vectors.
2. They are normal with $U^\dagger = U^{-1}$, so $UU^\dagger = U^\dagger U = I$

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:

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

The Hadamard 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:

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

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:

$$\sigma_X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

An important gate that arbitrarily rotates qubits by a phase shift θ without flipping bits is the phase gate P. This gate can also be understood to be a generalization of the Z gate and has the following matrix representation:

$$P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

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:

$$R_X(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}, \; R_Y(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}, \; R_Z(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

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 X or Z operation. Their matrices are shown below:

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

Another relevant multi-qubit gate for QAOA is the $U_{ZZ}(\theta)$ gate. This gate applies a phase to qubits based on the strength of their correlations and is useful for modeling costs between the two-body interactions present

in QUBO models. Mathematically, it applies a phase θ (rotates it by θ) when both qubits are 1 (in the $Z \otimes Z$ state). Thus, it is given by the equation $U_{ZZ}(\theta) = e^{i\theta(Z \otimes Z)}$ or the following matrix representation:

$$U_{ZZ}(\theta) = \begin{pmatrix} e^{i\theta} & 0 & 0 & 0 \\ 0 & e^{-i\theta} & 0 & 0 \\ 0 & 0 & e^{-i\theta} & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix}$$

There exist other single and multi-qubit quantum gates, however, these are the only ones essential to this tutorial.

Theoretically, it has been proven that there exist universal gate sets S (i.e, that any unitary transformation on an arbitrary number of qubits can approximated to finite precision using only finite sequence gates from S). Note as there are uncountably many unitary transformations and only countably infinite finite sequences of gates from a finite set, it is impossible to express all transformations without approximations. One such universal gate set is $\{R_X(\theta), R_Y(\theta), R_Z(\theta), CX, P(\theta)\}$ (Williams 2011). Universal gate sets are an open field of study that can be further explored in the following references (Sawicki et al. 2022), (Aharonov 2003).

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:

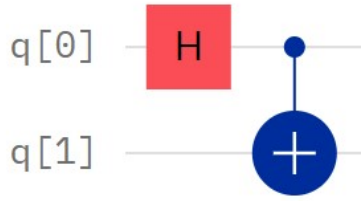


Figure 3 **First Bell state equivalent to** $|\phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$.

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

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

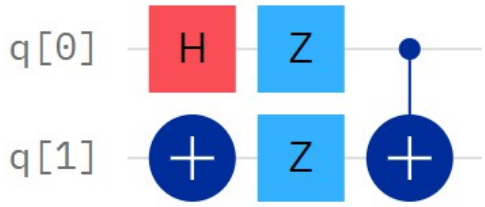


Figure 4 **Fourth Bell state equivalent to** $|\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$.

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

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

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 because of the non-commutativity of matrix multiplication. While the matrices in each "column" are tensored from bottom to top, this can be changed from the commutative property of tensor multiplication.

5.1.3. 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, in an arbitrary entangled state $|\psi\rangle \in \mathscr{H}_1 \otimes \mathscr{H}_2$, we cannot write

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

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

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 would also be 1. This property has been experimentally verified and offers potential for quantum speed-up.

5.1.4. 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 the 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:

$$M_{|0\rangle} = |0\rangle \langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$M_{|1\rangle} = |1\rangle \langle 1| = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Based on the measurement gates $M_{|b\rangle}$ and the quantum state $|\psi\rangle$. You can measure state m in the basis state $|b\rangle$ with probability

$$P(|\psi\rangle) = \langle\psi| M_{|b\rangle}^\dagger M_{|b\rangle} \, |\psi\rangle$$

5.1.5. NISQ In the realm of gate-based quantum computing, current technology is constrained to the Noisy Intermediate-Scale Quantum (NISQ) era. As indicated by the term "NISQ", these processors are limited in both size and accuracy (Brooks 2019). For example, IBM's Osprey, the most powerful publicly available quantum computer to date, has only 433 qubits, which restricts the range of problems it can solve. As of 2024, only two gate-based processors and three annealing-based processors have more than 1000 qubits.

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 and suffer from decoherence (loss of quantum properties) and quantum gate imperfections due to environmental influence. For these reasons, only circuits with low quantum depth (a few layers of quantum gates) have even moderate levels of accuracy. Although there is active research on promising error correction schemes, they are not yet applicable to larger-scale devices that prevent 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 advantage; quantum computers solve tasks that are unfeasible for classical computers.

5.2. Adiabatic Quantum Computing and Annealing

Adiabatic Quantum Computing (AQC) is an alternative form of quantum computing that uses the adiabatic theorem [5]. Currently, AQC devices are solely implemented in the form of D-Wave's Quantum Annealers (QA). AQC uses the continuous evolution of a quantum system, and although it is theoretically universal (Albash and Lidar 2018), Quantum Annealers can only be used to solve sampling or optimization problems.

Quantum Annealers, like many gate-based computers, also use superconducting qubits. However, they operate these qubits in an analog fashion, in contrast to gate-based quantum computers, which manipulate qubits digitally. As a result, Quantum Annealers can scale to thousands of qubits, whereas gate-based quantum devices are currently limited to just over 1,000 qubits.

Annealing algorithms are metaheuristic techniques for optimization problems inspired by the physical process of annealing in metallurgy, where metals are heated and then gradually cooled to remove defects and improve their properties.

To better appreciate the nuances of quantum annealing, we will first delve into simulated annealing. This classical technique will provide a foundation for understanding how its quantum counterpart builds on and improves upon these principles.

[5] The Adiabatic Theorem will be further explained in section 5.2

5.2.1. Simulated Annealing Simulated annealing (SA) is a probabilistic algorithm designed to find heuristic solutions to optimization problems. The method draws inspiration from the physical annealing process by gradually reducing the "temperature" of the system to locate a low-energy or optimal configuration.

SA begins by mapping the target problem to a cost function. This function maps possible solutions to their costs (also known as energies) to evaluate the quality of the solution. For the two solutions, the one with the lower energy is considered better.

The algorithm first initializes with a random solution and iteratively refines it to approximate a global optimum. In each iteration, the algorithm generates a neighboring solution by applying small perturbations to the current solution and evaluates its energy using the cost function. If the neighbor has a lower energy, it replaces the current solution. However, SA stands out from other metaheuristic methods by employing a **Temperature Schedule** and **Cooling Process** to determine whether to accept a neighboring solution, which makes it more resilient to getting stuck in local minima.

Temperature Schedule The likelihood of replacing the current solution with a neighbor of higher energy depends on the temperature T, which is initialized at the beginning of the algorithm. The probability of acceptance is given by:

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

where $f(x')$ and $f(x)$ are the energies of the neighboring and current solutions, respectively. This acceptance probability is derived from thermal annealing and governs the exploration process. When T is high or the energy difference $\Delta E := f(x') - f(x)$ is small, the algorithm is more likely to accept worse solutions, promoting exploration. Over time, T decreases, reducing the probability of such moves and encouraging the exploitation of promising regions. A key limitation of SA is its reduced ability to explore solutions when ΔE is large.

Cooling Process The cooling process reduces the temperature T at each iteration, typically using an exponential decay governed by a cooling factor $0 < \alpha < 1$:

$$T_{\text{new}} = \alpha \cdot T_{\text{old}}.$$

As T decreases, the algorithm becomes less likely to accept suboptimal solutions, progressively shifting its focus to exploiting the current solution. This gradual reduction ensures the convergence to more optimal solutions at the end of the process.

Simulated annealing offers several advantages, including its ability to escape local minima (Rutenbar 1989), simplicity in implementation, and adaptability to various problem domains. However, it also has limitations, such as its reliance on parameter tuning (e.g., initial temperature, cooling factor), slower convergence compared to other methods (Cohn and Fielding 1999), and suboptimal performance when there is a large energy difference (ΔE) (de Falco and Tamascelli 2011).

Require: Objective function $f(x)$, initial solution x, initial temperature T_{initial}, final temperature T_{final}, cooling factor α, and maximum iterations per temperature N_{iter}.

Ensure: Approximate solution x_{best} minimizing $f(x)$.

procedure SIMULATEDANNEALING(f, x, T_{initial}, T_{final}, α, N_{iter})
  $T \leftarrow T_{\text{initial}}$ ▷ Initialize temperature
  $x_{\text{best}} \leftarrow x$ ▷ Initialize the best solution with the initial solution
  while $T > T_{\text{final}}$ **do** ▷ Iterate until the system cools down
    for $i = 1$ to N_{iter} **do** ▷ Iterate for a fixed number of steps at each temperature
      $x' \leftarrow$ Generate a random neighbor of x ▷ Perturb the current solution
      $\Delta E \leftarrow f(x') - f(x)$ ▷ Evaluate the energy difference
      if $\Delta E < 0$ **then** ▷ If the neighbor is better, accept it
        $x \leftarrow x'$
        if $f(x) < f(x_{\text{best}})$ **then** ▷ Update the best solution if needed
          $x_{\text{best}} \leftarrow x$
      else
        $p_{\text{accept}} \leftarrow e^{-\Delta E/T}$ ▷ Calculate acceptance probability
        if $\text{rand}(0,1) < p_{\text{accept}}$ **then** ▷ Accept inferior solution probabilistically
          $x \leftarrow x'$
    $T \leftarrow \alpha \times T$ ▷ Reduce temperature based on the cooling factor
  return x_{best} ▷ Return the best solution found

5.2.2. Quantum Annealing Quantum annealing (QA) is a prominent example of adiabatic quantum computing, closely related to simulated annealing. It provides a theoretical advantage over classical algorithms by exploiting quantum mechanical principles, particularly quantum tunneling. Unlike many quantum computing techniques, quantum annealing is highly specialized for optimization problems and is implemented on quantum annealing platforms like D-Wave's quantum processors.

In quantum annealing, the optimization problem is first transformed into an objective function, akin to the cost function used in simulated annealing. This function maps possible solutions to their associated energy values, with the lowest energy corresponding to the optimal solution. The objective function is then encoded into a problem Hamiltonian, a quantum operator that represents the energy of the system. Understanding the Hamiltonian is crucial for both quantum annealing and the quantum approximate optimization algorithm (QAOA), so we will provide a brief explanation of its role in quantum mechanics.

Hamiltonian The Hamiltonian H is a quantum-mechanical operator that describes the total energy of a system. It has several key properties:

- It is Hermitian ($H^\dagger = H$), ensuring that its eigenvalues are real.

- Its eigenvalues represent the possible energy levels of the system, while the corresponding eigenvectors describe the quantum states.
- It governs the time evolution of the system's quantum state $|\psi(t)\rangle$ via the Schrödinger equation, which is essential in quantum annealing:

$$i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = H|\psi(t)\rangle.$$

In quantum annealing, two Hamiltonians are used: the initial (driver) Hamiltonian and the cost Hamiltonian.

- **Driver Hamiltonian (H_D):** The driver Hamiltonian is designed such that its ground state is easy to prepare, typically placing all qubits in a uniform superposition. It is given by:

$$H_D = -\sum_i \sigma_x^{(i)},$$

where $\sigma_x^{(i)}$ are the Pauli-X operators acting on the i-th qubit.

- **Cost Hamiltonian (H_C):** The cost Hamiltonian encodes the optimization problem, representing the energy of the quantum system for the given problem. Finding the ground state of H_C is typically NP-Hard, making it challenging for classical computers to solve.

These two Hamiltonians (H_D and H_C) are used in quantum annealing, which also relies on the adiabatic theorem for time evolution.

Adiabatic Theorem. *Let $H(s) = (1-s(t))H_D + s(t)H_C$ be a time-dependent Hamiltonian that smoothly evolves from the Driver Hamiltonian H_D to the cost Hamiltonian H_C over a monotonic function $s(t)$ in the interval $t \in (0,T)$. If the system starts in the ground state of H_D, and the evolution is sufficiently slow such that the total time T satisfies the adiabatic condition:*

$$T \gg \frac{1}{\Delta_{\min}^2},$$

where $\Delta_{\min}$ is the minimum energy gap between the ground state and the first excited state of $H(s)$, the system will remain in the ground state of $H(s)$ throughout the evolution. At $s(t) = 1$, the system will end in the ground state of H_C, which corresponds to the optimal solution of the problem.

With these Hamiltonians and the adiabatic theorem, the quantum annealing process proceeds as follows:

The D-Wave annealer receives and solves an Ising Hamiltonian of the form:

$$H_{\text{Ising}} = -\frac{A(s)}{2}\underbrace{\left(\sum_i \sigma_X^{(i)}\right)}_{\text{Initial Hamiltonian}} + \frac{B(s)}{2}\underbrace{\left(\sum_i h_i\sigma_Z^{(i)} + \sum_{i>j} J_{ij}\sigma_Z^{(i)}\sigma_Z^{(j)}\right)}_{\text{Final Hamiltonian}}, \tag{2}$$

where σ_i are the Pauli matrices acting on qubit q_i, and h_i and J_{ij} are the biases and couplings, respectively. As seen from the equation, the Ising Hamiltonian consists of an initial and a final Hamiltonian.

Schrödinger Time Evolution The Ising Hamiltonian begins with $A(s)$ being a very large negative value and $B(s)$ being a small positive value. As a result, the physical system starts in the ground state of the initial Hamiltonian (or in a uniform superposition). During the annealing process, $A(s)$ gradually decreases in magnitude while $B(s)$ increases. The magnetic biases and couplings guide the physical system towards the ground state of the problem Hamiltonian. At the end of the anneal, the qubits collapse to the ground-state energy of the problem Hamiltonian, which represents the optimal solution.

The adiabatic theorem is crucial to the success of quantum annealing, as it ensures that the final physical system is in its ground state (i.e., the optimal solution is obtained) if the evolution is slow enough.

Algorithm 1 Quantum Annealing

procedure QUANTUM_ANNEALING($|\psi_0\rangle$, $A(s)$, $B(s)$, h_i, J_{ij}, T)
  $s \leftarrow 0$ ▷ Initialize annealing parameter
  $|\psi(s)\rangle \leftarrow |\psi_0\rangle$ ▷ Initial state
  best_solution $\leftarrow |\psi(s)\rangle$ ▷ Track best solution
  while $s < 1$ **do** ▷ Loop until final annealing step
    Compute $H_{\text{Ising}}(s)$:

$$H_{\text{Ising}}(s) = -\frac{A(s)}{2}\sum_i \sigma_X^{(i)} + \frac{B(s)}{2}\left(\sum_i h_i \sigma_Z^{(i)} + \sum_{i>j} J_{ij}\sigma_Z^{(i)}\sigma_Z^{(j)}\right)$$

    Evolve state $|\psi(s)\rangle$ under the Schrödinger equation for Δs using $H_{\text{Ising}}(s)$:

$$i\hbar\frac{d}{ds}|\psi(s)\rangle = H_{\text{Ising}}(s)|\psi(s)\rangle$$

    Update $s \leftarrow s + \Delta s$
  best_solution $\leftarrow$ Measure the ground state $|\psi(s)\rangle$ ▷ Final state corresponds to optimal solution
  return best_solution

Quantum Tunneling During time evolution, the quantum system could encounter energy barriers where the differences in energy between the current state and neighboring states are very large. Simulated annealing often struggles with finding the global optimum in these circumstances, due to its acceptance probability function, however, quantum annealing prevails due to Quantum Tunneling (SHIFMAN 2002). Because of the wavelike properties of the encoded Hamiltonians, there is a non-zero probability that the system can tunnel through the energy barrier avoid local optima and find global optima. This phenomenon is the main quantum-mechanical advantage of QA.

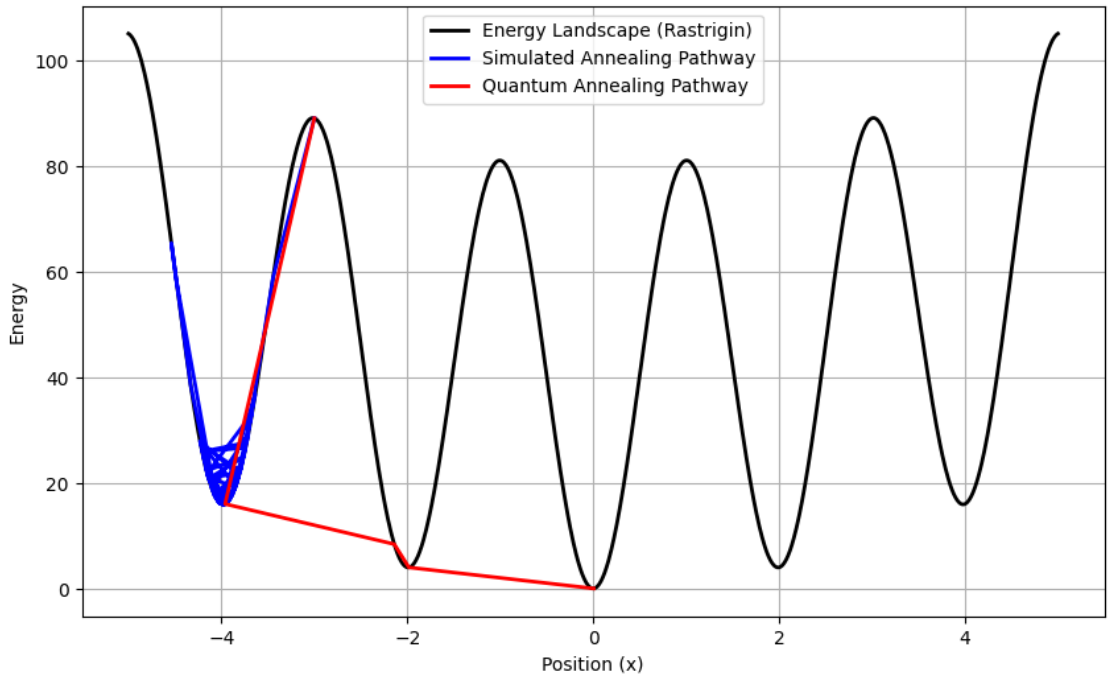


Figure 5 **Simulated Annealing vs Quantum Annealing Optimization Pathway over Rastrigin ($A = 40, x_i = -3$)**

5.3. Quantum Approximate Optimization Algorithm (QAOA)

The Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al. 2014) is a quantum-classical algorithm that is used to solve combinatorial optimization problems on gate-based quantum computers. At a high level, it can be viewed as a discretized version of the adiabatic evolution of quantum annealing. Unlike many other quantum algorithms, QAOA has a low circuit depth, which makes it less susceptible to quantum decoherence. As a result, QAOA is currently one of the promising NISQ algorithms for problems in finance, operations research, and logistics.

Its more famous counterpart, Variational Quantum Eigensolver (VQE), works similarly, however, was designed to solve eigenvalue problems. VQE is best known for its applications in computational chemistry, such as finding the ground state energy of a molecule (Peruzzo et al. 2014) (finding the smallest eigenvalue of the molecular Hamiltonian). Due to the similarity between QAOA and VQE, much of the information that describes QAOA can also be applied to VQE.

5.3.1. Problem Mapping The first step in implementing QAOA is to map the optimization problem to an error function f. This function is analogous to the objective functions used in simulated annealing and quantum annealing. It takes a candidate solution vector for the optimization problem and returns a value that quantifies how far the solution is from the optimal one, essentially representing the ”error” of the solution. Similarly, as simulated annealing and quantum annealing select solutions based on minimizing the energy associated with their objective functions, in QAOA, the goal is to identify the solutions that minimize the error function f, thus finding the optimal solution.

This function is crucial for evaluating the performance of the QAOA circuit. Since quantum circuits operate probabilistically, the algorithm does not just evaluate the function for a single input; instead, it computes the expected value of the objective function across a set of potential solutions. Lower expected values correspond to higher-quality solutions.

The algorithm begins by initializing random parameters $\{\beta\}_{i\geq 0}$ and $\{\gamma\}_{i\geq 0}$, which are then fed into the initial parametrized circuit. The circuit generates an initial solution that is then evaluated using the objective function.

5.3.2. Creating the Parametrized Circuit A parametrized quantum circuit is created by creating a layer of Hadamard gates and then using multiple alternating mixer and cost Operator layers.

The Hadamard H gates are used to convert the initial basis state $|0\rangle$ into a uniform superposition. $|+\rangle := \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ to explore all configurations.
The algorithm then alternates between two key operators: the mixer operator and the cost operator.

1. The mixer Operator $e^{-i\beta H_M}$ is created from the parameters passed in $\{\beta\}_{i\geq 0}$ and a mixer Hamiltonian H_M (equivalent to the driver Hamiltonian from quantum annealing) generated from the original QUBO problem. This operator is used to explore the search space more efficiently by encouraging transitions between different bases.

2. The cost Operator $e^{-i\gamma H_C}$ encodes the objective function of the initial optimization problem and is created from the passed in $\{\gamma\}_{i\geq 0}$ and a cost Hamiltonian H_C (the same as the cost Hamiltonian from quantum annealing) generated from the initial QUBO problem. This operator is used to guide the system towards states with more optimal solutions.

The number of times the mixer and cost operators are cycled depends on a parameter p. This circuit is parametrized because it depends on $2p$ parameters γ and β for the cost and mixer operators. Together, these components make up the parametrized circuit (also known as ansatz).

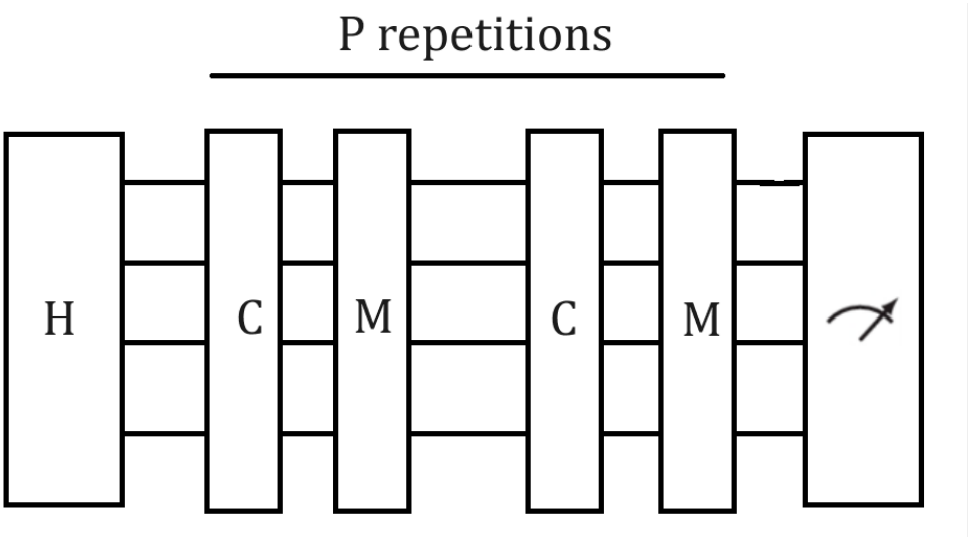


Figure 6 Generic QAOA Circuit Framework

5.3.3. Classical Optimization A classical optimizer then uses the outputs to refine the β and γ parameters of the ansatz to minimize the error function. As mentioned earlier, this error function would compute the expected error given the ansatz parameters. Since the expectation function is nonconvex, it is an active research topic on which optimizers are best (Mazumder et al. 2024), (Pellow-Jarman et al. 2024) however, heuristics and gradient-based optimization methods are commonly employed. Once optimization is done, the circuit is measured and an approximate solution is obtained. The complete pseudocode for QAOA is shown below:

Algorithm 2 Quantum Approximate Optimization Algorithm (QAOA)

procedure QAOA(cost Hamiltonian H_C, Mixing Hamiltonian H_M, Number of Cycles p, Initial Parameters $\vec{\beta}, \vec{\gamma}$)
  Initialize state $|\psi\rangle = |+\rangle^n$ ▷ Start with all qubits in an equal superposition state
  Initialize parameters $\vec{\beta}, \vec{\gamma} \in \mathbb{R}^p$ randomly ▷ Set initial values for parameters
  while not converged **do**
    for $i = 1$ to p **do** ▷ Loop over each cycle
      Apply the mixer Hamiltonian: $e^{-i\beta_i H_M}$ ▷ Use R_X gates to implement
      Apply the cost Hamiltonian: $e^{-i\gamma_i H_C}$ ▷ Use controlled ops. or U_{ZZ} gates to implement
    Measure the state $|\psi\rangle$ to estimate the cost function value ▷ Evaluate the solution's quality
    Update the parameters $\vec{\beta}, \vec{\gamma}$ using Classical Optimizer ▷ e.g., gradient descent to minimize cost
  Measure the final state $|\psi\rangle$ ▷ Obtain the final result after convergence
  return Measurement outcome (approximate solution) ▷ Return the best solution found

The exact means to obtain mixer and cost Hamiltonians for arbitrary QUBOs to construct mixer and cost Operators will be discussed in more detail in Section 6.

6. Solving QUBOs on Quantum Computers

Three algorithms for five previously mentioned problems are implemented in the following repo: Quantum Approximate Optimization Algorithm (QAOA) (Farhi et al. 2014), Quantum Annealing (QA) (de Falco and Tamascelli 2011), and simulated annealing (SA) (Kirkpatrick et al. 1983b). QAOA is implemented in three different ways in IBM quantum back-ends and simulators: Vanilla QAOA (Notebook 1), Open-QAOA (Notebook 2), and Qiskit (Notebook 3). In this paper, we provide code snippets for implementing the Quantum Approximate Optimization Algorithm (QAOA) using IBM's QASM Simulator. For readers interested in applying QAOA to actual IBM quantum processors, we also provide similar implementations for the IBM backends "IBM Kyiv," "IBM Brisbane," and "IBM Sherbrooke," all of which use the Eagle r3 Processor with 127 qubits. Specifically, these 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 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[6]. We provide a brief description of each implementation's pros and cons at the end of this section, however, further details can be found in the corresponding subsections.

6.1. Vanilla QAOA

Regardless of the problem addressed, the vanilla implementation of QAOA requires a cost Hamiltonian and a mixer Hamiltonian. These Hamiltonians are combined to create a QAOA circuit. All features were implemented following the instructions provided by the Qiskit textbook (Abbas et al. 2020). This first example focuses on the Number Partitioning problem to illustrate the steps and the code.

6.1.1. Constructing Operators To create the cost and mixer Operators, we need the tunable parameters γ, β and the cost and mixer Hamiltonians of the problem. In the following section, we will explain how to obtain each of these structures and how to use them to create the corresponding operator.

First, note that the parameters γ, β are arbitrarily initialized. In this example, they are each initialized as 0.5. These parameters will be adjusted during the optimization phase for the QAOA circuit.

Second, we aim to obtain the Hamiltonians. For an arbitrary QUBO in the form of

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

we get the corresponding cost Hamiltonian as

$$H_C = \sum_{i,j=1}^{n} \frac{1}{4} Q_{ij} Z_i Z_j - \sum_{i=1}^{n} \frac{1}{2} \left(c_i + \sum_{j=1}^{n} Q_{ij} \right) Z_i$$

so the corresponding cost Operator is

$$e^{-i\gamma H_C} = \prod_{i,j=1}^{n} R_{Z_i Z_j} \left(\frac{1}{4} Q_{ij} \gamma \right) \prod_{i=1}^{n} R_{Z_i} \left(\frac{1}{2} \left(c_i + \sum_{j=1}^{n} Q_{ij} \right) \gamma \right)$$

On a circuit with n qubits (circuit width of n-qubits), this operator can be constructed by applying two layers of gates. The first layer is a layer of R_Z gates to each qubit i (for $i = 1, 2, \ldots, n$), with angles θ_{ij1}:

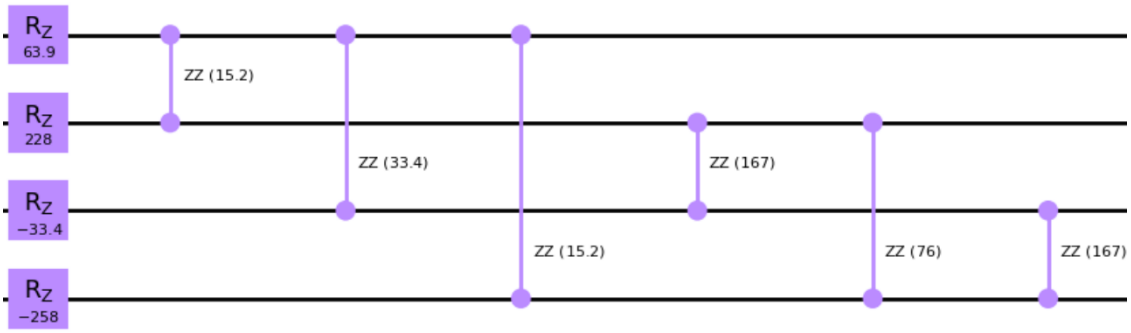
$$\theta_{ij1} = \frac{\gamma}{2} (c_i + \sum_{i=1}^{n} Q_{ij})$$

This layer of R_Z gates is followed by a layer of U_{ZZ} gates applied on each pair of qubits i and i at angles θ_{i2}:

$$\theta_{i2} = \frac{1}{4} Q_{ij} \gamma.$$

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

```
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
```



Similarly, the mixer operator depends on β and the mixer Hamiltonian.

The mixer Hamiltonian is

$$H_M = \sum_i^n \sigma_{X_i}$$

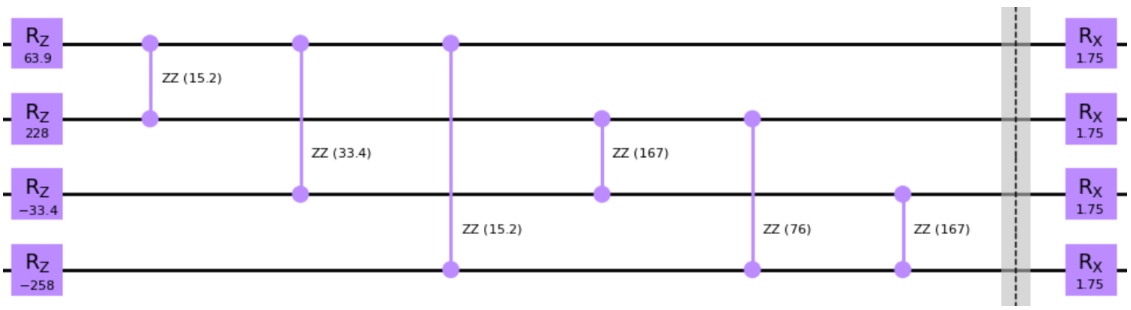
so the corresponding mixer operator is

$$e^{-i\beta H_M} = \prod_{i=1}^n R_X(2\beta)$$

This mixer operator can be applied by applying R_X gates to each qubit i (for $i = 1, 2, \ldots, n$) at angle θ_{i3}:

$$\theta_{i3} = 2\beta.$$

```
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
```

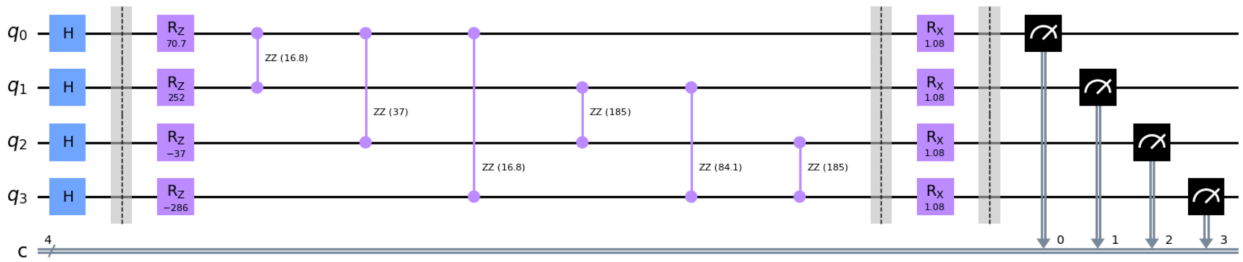


6.1.2. QAOA Circuit Once the generic framework is established, the quadratic and linear coefficients of the Number Partitioning QUBO are substituted. The model is created using DOcplex (Cplex 2009), converted to a QUBO, and partitioned into quadratic and linear terms for gate implementation. The test example uses the set $\{1,5,5,11\}$ for partitioning.

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

```
arr = [1, 5, 11, 5]
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 parameters γ and β are optimized using a classical optimizer. To optimize, we define an objective function and an expectation function for the output of the circuit. The objective function measures the quality of the 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.

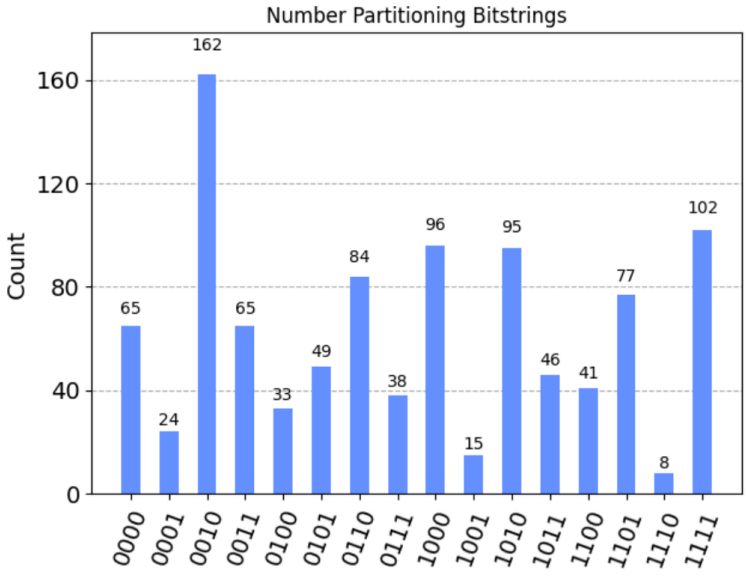


Figure 7 Sampled QAOA results for the Number Partitioning QUBO

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

6.2. OpenQAOA

OpenQAOA (Sharma et al. 2022) 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 (Karalekas et al. 2020), Amazon Braket (Amazon Web Services 2020), and Microsoft Azure (Hooyberghs 2022). The parent company of OpenQAOA also provides usage of custom simulators.

We illustrate the implementation of the *Max-Cut* problem. This process requires the same steps as for the vanilla implementation of QAOA. However, most of the technical challenges are solved by the OpenQAOA API. Like 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)
```

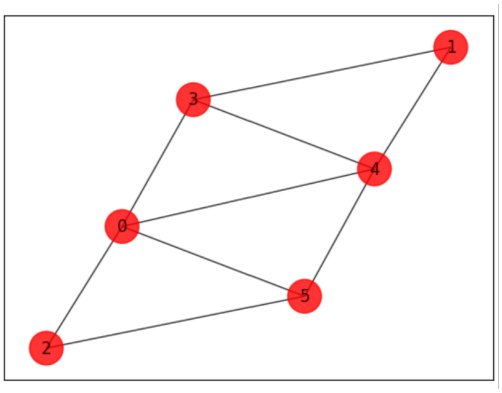


Figure 8 **Generated Max-Cut Graph Instance**

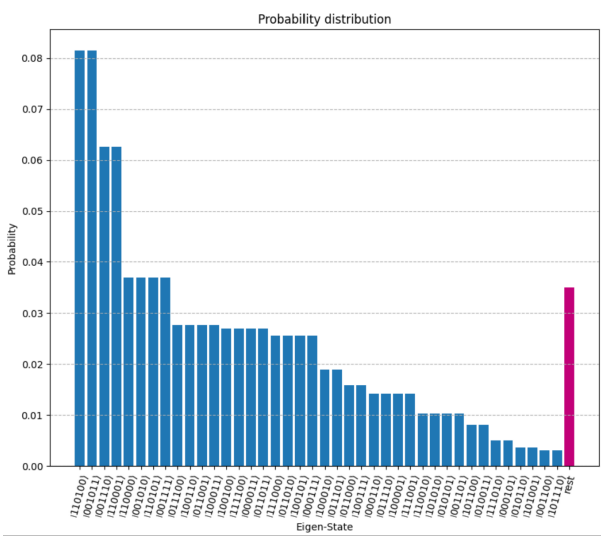
The QUBO creation process for Max-Cut is very similar to the process with Number Partitioning.

```
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
```

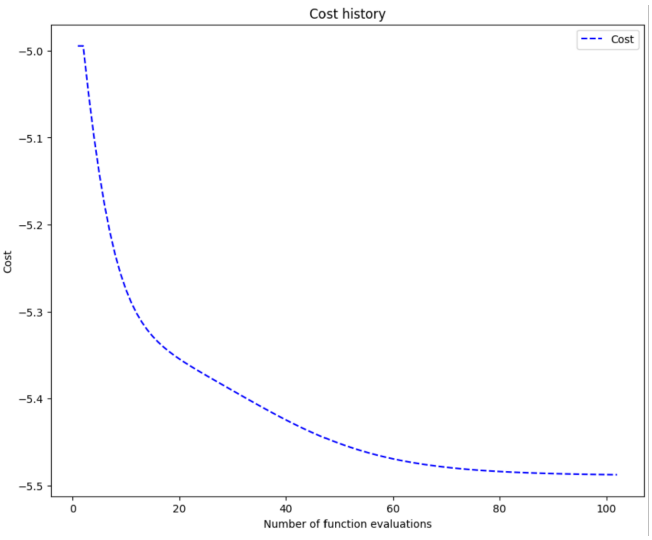
6.2.1. OpenQAOA Circuit Once the QUBO is created, OpenQAOA provides a short and easy implementation to create the QAOA circuit. To start, you only need to initialize the QAOA object (`q = QAOA()`) and then define its properties. These properties include whether the circuit will run locally on a simulator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

6.2.2. 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

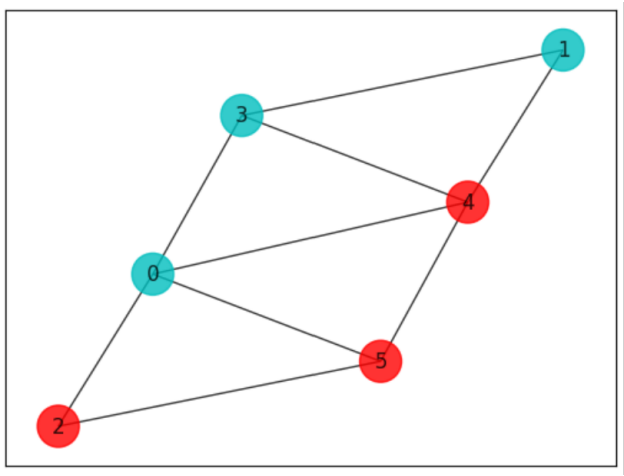


Figure 10 Optimal Graph Partitioning (Max Cut) based on Most Common Bitstring: (Nodes 2, 4 and 5) and Nodes (0, 1, 3)

6.3. QAOA via Qiskit

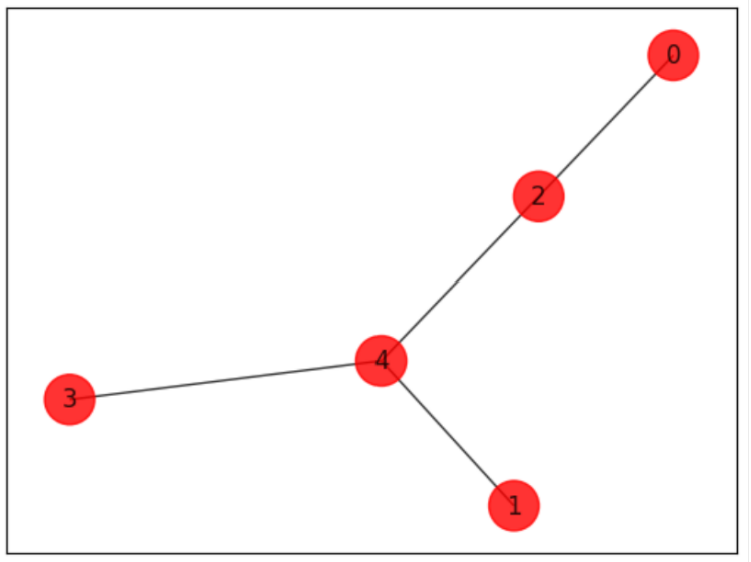
Qiskit-Optimization is a part of IBM Qiskit's open-source quantum computing framework. It provides a wide array of algorithms and built-in application classes for many canonical problems.

Qiskit Optimization is an extremely useful and easy way to solve QUBOs with various quantum algorithms including Variational Quantum Eigensolver (Peruzzo et al. 2014), Adaptive Grover (Gilliam et al. 2021), 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. The key limitations of Qiskit optimization are its extremely long optimization times and a limited set of optimizers.

Let us now consider the *Minimum Vertex Cover* problem. Similarly 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 OpenQAOA, we just need to define a QAOA object. This object takes a sampler, optimizer, number of repetitions, 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.

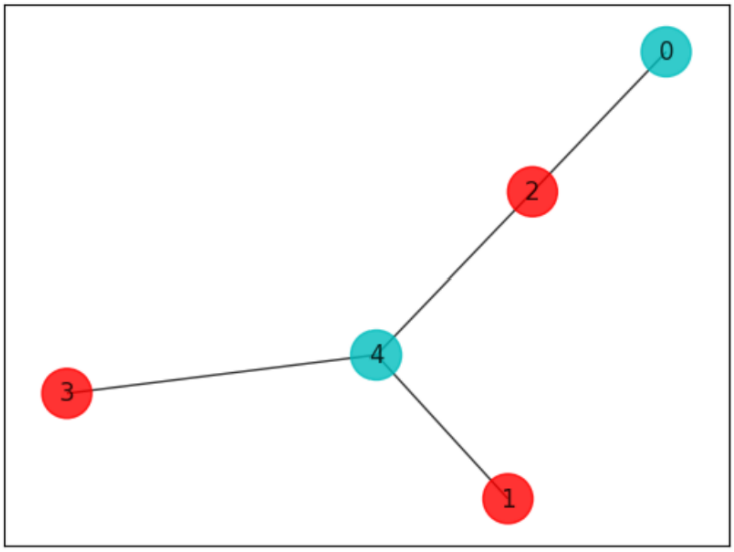


Figure 11 Optimal Minimum Vertex Cover: Nodes 0 and 4

6.4. Simulated Annealing

Let us now turn to 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 custom to this paper, 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 binary quadratic model (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 an equilibrium more efficiently (although potentially not optimal). Like 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)
```

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 repository. 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 12 Solution to Order Partitioning instance

6.5. Quantum Annealing

The final algorithm explored is quantum annealing, implemented using D-Wave's Quantum Annealer for the cancer genomic application. With access to significantly more qubits (a few thousand compared to IBM's 127 qubits), D-Wave enables the execution of larger problem instances.

The cancer genomic problem requires significant preprocessing effort. First, to construct the QUBO cancer pathway, the disease data must be read from some external source. For this tutorial, data were collected from The Cancer Genome Atlas Acute Myeloid Leukemia dataset posted on cBioPortal (Ley et al. 2013).

```
# 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 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 13 Patient-Gene Dictionary sample

Note that with this dictionary, both the patients and genes are enumerated. 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 all the patients.

```
# Initialize diagonal matrix
D = np.zeros((n, n))

# Populate diagonal matrix
for i, gene in enumerate(geneList):
    D[i, i] = sum(gene in genes for genes in PatientGeneDict.values())
```

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 in a list and then use a dictionary to map each patient to its list of gene pairs. Then this new patient-gene pair dictionary is used to create matrix $\mathbf{A}$ where the index A_{ij} represents the number of patients who possess gene pair (i, j). Note that 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])
```

```
A = np.zeros((n, n))

for i in range(n):
    for j in range(i + 1, n):  # iterate over upper triangular part
        count = sum((geneList[i], geneList[j]) in pairs for pairs in
           PatientGeneDictPairs.values())
        A[i, j] = A[j, i] = count  # Exploit symmetry
```

Once the $\mathbf{D}$ and $\mathbf{A}$ matrices are constructed, we create the QUBO model and later the 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 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 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 14 Cancer Gene Pathway discovered through quantum annealing

7. 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 the 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) (Sodhi and Tayur 2023). To take better advantage of these nascent devices, it is necessary to develop decomposition methods, an active area of research (Tayur and Tenneti 2024), and a topic for future tutorial.

Implementation	**Characteristics**
QAOA (Vanilla)	• High circuit customization • High optimizer customization • Difficult implementation • Accuracy substantially decreases with problem size
QAOA (OpenQAOA)	• Moderate circuit customization • Moderate optimizer customization • Easy implementation • Best time vs. accuracy tradeoff among implementations
QAOA (Qiskit)	• Low circuit customization • Low optimizer customization • Easy implementation • Time exponentially increases with problem size
Simulated Annealing	• High parameter customization • Handles larger problem sizes • Fastest runtime • Most accurate results • Easy implementation
Quantum Annealing	• High topology customization (not discussed in this paper) • Handles larger problem sizes • Fastest quantum runtime • Most accurate quantum results • Easy implementation

Table 1 Comparison of Various Implementations and Their Characteristics

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

References

Abbas A, Andersson S, Asfaw A, Corcoles A, Bello L, Ben-Haim Y, Bozzo-Rey M, Bravyi S, Bronn N, Capelluto L, Vazquez AC, Ceroni J, Chen R, Frisch A, Gambetta J, Garion S, Gil L, Gonzalez SDLP, Harkins F, Imamichi T,

Jayasinha P, Kang H, h Karamlou A, Loredo R, McKay D, Maldonado A, Macaluso A, Mezzacapo A, Minev Z, Movassagh R, Nannicini G, Nation P, Phan A, Pistoia M, Rattew A, Schaefer J, Serrano DE, Shabani J, Smolin J, Stenger J, Temme K, Tod M, Wanzambi E, Wood S, Wootton J (2020) Learn quantum computation using qiskit. URL https://qiskit.org/textbook/.

Abello J, Butenko S, Pardalos PM, Resende MG (2001) Finding independent sets in a graph using continuous multivariable polynomial formulations. *Journal of Global Optimization* 21(2):111–137, ISSN 0925-5001, URL http://dx.doi.org/10.1023/a:1011968411281.

Aharonov D (2003) A simple proof that toffoli and hadamard are quantum universal. URL https://arxiv.org/abs/quant-ph/0301040.

Albash T, Lidar DA (2018) Adiabatic quantum computation. *Reviews of Modern Physics* 90(1), ISSN 1539-0756, URL http://dx.doi.org/10.1103/revmodphys.90.015002.

Alghassi H, Dridi R, Robertson AG, Tayur S (2019) Quantum and quantum-inspired methods for de novo discovery of altered cancer pathways. *bioRxiv* URL http://dx.doi.org/10.1101/845719.

Alidaee B, Glover F, Kochenberger G, Rego C (2005) A new modeling and solution approach for the number partitioning problem. *JAMDS* 9:113–121, URL http://dx.doi.org/10.1155/JAMDS.2005.113.

Amazon Web Services (2020) Amazon Braket. URL https://aws.amazon.com/braket/.

Bian Z, Chudak F, Israel RB, Lackey B, Macready WG, Roy A (2016) Mapping constrained optimization problems to quantum annealing with application to fault diagnosis. *Frontiers in ICT* 3, ISSN 2297-198X, URL http://dx.doi.org/10.3389/fict.2016.00014.

Brooks M (2019) Beyond quantum supremacy: the hunt for useful quantum computers. *Nature* 574(7776):19–21, ISSN 1476-4687, URL http://dx.doi.org/10.1038/d41586-019-02936-3.

Chen J, Kanj IA, Xia G (2006) Improved parameterized upper bounds for Vertex Cover. *Mathematical Foundations of Computer Science 2006*, 238–249 (Berlin, Heidelberg: Springer Berlin Heidelberg), ISBN 978-3-540-37793-1.

Cohn H, Fielding M (1999) Simulated annealing: Searching for an optimal temperature schedule. *SIAM J. on Optimization* 9(3):779–802, ISSN 1052-6234, URL http://dx.doi.org/10.1137/S1052623497329683.

Commander CW (2009) *Maximum cut problem, MAX-CUTMaximum Cut Problem, MAX-CUT*, 1991–1999 (Boston, MA: Springer US), ISBN 978-0-387-74759-0, URL http://dx.doi.org/10.1007/978-0-387-74759-0_358.

Cplex II (2009) V12. 1: User's manual for cplex. *International Business Machines Corporation* 46(53):157.

de Falco D, Tamascelli D (2011) An introduction to quantum annealing. *RAIRO - Theoretical Informatics and Applications* 45, URL http://dx.doi.org/10.1051/ita/2011013.

Farhi E, Goldstone J, Gutmann S (2014) A quantum approximate optimization algorithm.

Febrianti W, Sidarto KA, Sumarti N (2023) Solving constrained mean-variance portfolio optimization problems using spiral optimization algorithm. *International Journal of Financial Studies* 11(1), ISSN 2227-7072, URL http://dx.doi.org/10.3390/ijfs11010001.

Gärtner B, Matoušek J (2012) *Semidefinite Programming*, 15–25 (Berlin, Heidelberg: Springer Berlin Heidelberg), ISBN 978-3-642-22015-9, URL http://dx.doi.org/10.1007/978-3-642-22015-9_2.

Gerges F, Zouein G, Azar D (2018) Genetic algorithms with local optima handling to solve sudoku puzzles. *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*, 19–22, ICCAI '18 (New York, NY, USA: Association for Computing Machinery), ISBN 9781450364195, URL http://dx.doi.org/10.1145/3194452.3194463.

Gilliam A, Woerner S, Gonciulea C (2021) Grover adaptive search for constrained polynomial binary optimization. *Quantum* 5:428, ISSN 2521-327X, URL http://dx.doi.org/10.22331/q-2021-04-08-428.

Ginsburgh V VPA (1969) Un algorithme de programmation quadratique en variables binaires. *RAIRO - Operations Research - Recherche Opérationnelle* 3(V2):57–73, URL http://eudml.org/doc/104473.

Glover F (1989) Tabu search—part i. *ORSA Journal on computing* 1(3):190–206.

Goemans MX, Williamson DP (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* 42(6):1115–1145, ISSN 1557-735X, URL http://dx.doi.org/10.1145/227683.227684.

Guerreschi GG, Matsuura AY (2019) Qaoa for max-cut requires hundreds of qubits for quantum speed-up. *Scientific Reports* 9(1), ISSN 2045-2322, URL http://dx.doi.org/10.1038/s41598-019-43176-9.

Hammer PL, Rudeanu S (1968) *Boolean Methods in Operations Research and Related Areas* (Springer Berlin Heidelberg), ISBN 9783642858239, URL http://dx.doi.org/10.1007/978-3-642-85823-9.

Helmberg C, Rendl F (1998) Solving quadratic (0, 1)-problems by semidefinite programs and cutting planes. *Mathematical Programming* 82(3):291–315, ISSN 1436-4646, URL http://dx.doi.org/10.1007/bf01580072.

Hooyberghs J (2022) *Azure Quantum*, 307–339 (Berkeley, CA: Apress), ISBN 978-1-4842-7246-6, URL http://dx.doi.org/10.1007/978-1-4842-7246-6_11.

Kadowaki T, Nishimori H (1998) Quantum annealing in the transverse ising model. *Physical Review E* 58(5):5355–5363, ISSN 1095-3787, URL http://dx.doi.org/10.1103/physreve.58.5355.

Karalekas PJ, Tezak NA, Peterson EC, Ryan CA, da Silva MP, Smith RS (2020) A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quantum Science and Technology* 5(2):024003, ISSN 2058-9565, URL http://dx.doi.org/10.1088/2058-9565/ab7559.

Karp RM (1972) *Reducibility among Combinatorial Problems*, 85–103 (Springer US), ISBN 9781468420012, URL http://dx.doi.org/10.1007/978-1-4684-2001-2_9.

Kikuura S, Igata R, Shingu Y, Watabe S (2024) Performance benchmarking of quantum algorithms for hard combinatorial optimization problems: A comparative study of non-ftqc approaches. URL http://dx.doi.org/10.48550/ARXIV.2410.22810.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983a) Optimization by simulated annealing. *Science* 220(4598):671–680, URL http://dx.doi.org/10.1126/science.220.4598.671.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983b) Optimization by simulated annealing. *Science* 220(4598):671–680, URL http://dx.doi.org/10.1126/science.220.4598.671.

Leiserson M, Blokh D, Sharan R, Raphael B (2013) Simultaneous identification of multiple driver pathways in cancer. *PLoS computational biology* 9:e1003054, URL http://dx.doi.org/10.1371/journal.pcbi.1003054.

Ley TJ, Miller CB, xia Ding L, Raphael BJ, Mungall AJ, Robertson AG, Hoadley KA, Triche TJ, Laird PW, Baty JD, Fulton LL, Fulton RS, Heath SE, Kalicki-Veizer JM, Kandoth C, Klco JM, Koboldt DC, Kanchi KL, Kulkarni S, Lamprecht TL, Larson DE, Lin L, Lu C, McLellan MD, McMichael JF, Payton JE, Schmidt HK, Spencer DH, Tomasson MH, Wallis JW, Wartman LD, Watson MA, Welch JSS, Wendl MC, Ally A, Balasundaram M, Birol I, Butterfield YSN, Chiu R, Chu A, Chuah E, Chun HJE, Corbett R, Dhalla N, Guin R, He A, Hirst C, Hirst M, Holt RA, Jones SP, Karsan A, Lee D, Li HI, Marra MA, Mayo M, Moore RA, Mungall KL, Parker JD, Pleasance ED, Plettner P, Schein JE, Stoll D, Swanson L, Tam A, Thiessen N, Varhol R, Wye NH, Zhao Y, Gabriel S, Getz G, Sougnez C, Zou L, Leiserson MDM, Vandin F, Wu HT, Applebaum FR, Baylin SB, Akbani R, Broom BM, Chen K, Motter TC, Nguyen K, Weinstein JN, Zhang N, Ferguson ML, Adams C, Black AD, Bowen J, Gastier-Foster JM, Grossman T, Lichtenberg TM, Wise L, Davidsen T, Demchok JA, Shaw KRM, Sheth M, Sofia HJ, Yang L, Downing JR, Eley GD (2013) Genomic and epigenomic landscapes of adult de novo acute myeloid leukemia. *The New England journal of medicine* 368 22:2059–74, URL https://api.semanticscholar.org/CorpusID:20280312.

Loke ZX, Goh SL, Kendall G, Abdullah S, Sabar NR (2023) Portfolio optimization problem: A taxonomic review of solution methodologies. *IEEE Access* 11:33100–33120, URL http://dx.doi.org/10.1109/ACCESS.2023.3263198.

Lucas A (2014) Ising formulations of many NP problems. *Frontiers in Physics* 2, URL http://dx.doi.org/10.3389/fphy.2014.00005.

Mazumder AR, Sen A, Sen U (2024) Benchmarking metaheuristic-integrated qaoa against quantum annealing. URL https://arxiv.org/abs/2309.16796.

McGeoch CC (2014) *Adiabatic Quantum Computation and Quantum Annealing* (Kenfield, CA: Morgan&Claypool), ISBN 978-1-62705-592-5.

Mertens S (1998) Phase transition in the number partitioning problem. *Phys. Rev. Lett.* 81:4281–4284, URL http://dx.doi.org/10.1103/PhysRevLett.81.4281.

Mertens S (2003) The easiest hard problem: Number partitioning.

Nielsen MA, Chuang IL (2011) *Quantum Computation and Quantum Information: 10th Anniversary Edition* (New York, NY, USA: Cambridge University Press).

Pardalos PM, Rodgers GP (1990) Computational aspects of a branch and bound algorithm for quadratic zero-one programming. *Computing* 45(2):131–144, ISSN 1436-5057, URL http://dx.doi.org/10.1007/bf02247879.

Pellow-Jarman A, McFarthing S, Sinayskiy I, Park DK, Pillay A, Petruccione F (2024) The effect of classical optimizers and ansatz depth on qaoa performance in noisy devices. *Scientific Reports* 14(1), ISSN 2045-2322, URL http://dx.doi.org/10.1038/s41598-024-66625-6.

Peruzzo A, McClean J, Shadbolt P, Yung MH, Zhou XQ, Love PJ, Aspuru-Guzik A, O'Brien JL (2014) A variational eigenvalue solver on a photonic quantum processor. *Nature Communications* 5(1), ISSN 2041-1723, URL http://dx.doi.org/10.1038/ncomms5213.

Preskill J (2018) Quantum computing in the nisq era and beyond. *Quantum* 2:79, ISSN 2521-327X, URL http://dx.doi.org/10.22331/q-2018-08-06-79.

Qiskit contributors (2023) Qiskit: An open-source framework for quantum computing. URL http://dx.doi.org/10.5281/zenodo.2573505.

Rieffel E, Polak W (2014) *Quantum Computing: A Gentle Introduction* (Cambridge, MA: The MIT Press), ISBN 978-0-262-526678.

Rutenbar R (1989) Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine* 5(1):19–26, URL http://dx.doi.org/10.1109/101.17235.

Sawicki A, Mattioli L, Zimborás Z (2022) Universality verification for a set of quantum gates. *Physical Review A* 105(5), ISSN 2469-9934, URL http://dx.doi.org/10.1103/physreva.105.052602.

Sharma V, Saharan NSB, Chiew SH, Chiacchio EIR, Disilvestro L, Demarie TF, Munro E (2022) Openqaoa – an sdk for qaoa.

SHIFMAN M (2002) QUANTUM TUNNELING. *Multiple Facets of Quantization and Supersymmetry*, 52–67 (WORLD SCIENTIFIC), URL http://dx.doi.org/10.1142/9789812777065_0012.

Siddhu V, Tayur S (2022) Five starter problems: Quantum information science via semi-definite programs. *Tutorials in Operations Research* 59–92.

Sodhi MS, Tayur SR (2023) Make Your Business Quantum-Ready Today. *Management and Business Review* .

Strash D (2016) On the power of simple reductions for the maximum independent set problem. Dinh TN, Thai MT, eds., *Computing and Combinatorics*, 345–356 (Cham: Springer International Publishing), ISBN 978-3-319-42634-1.

Tayur S, Tenneti A (2024) Quantum Annealing Research at CMU: Algorithms, Hardware, Applications. *Frontiers of Computer Science* .

Vandin F, Upfal E, Raphael BJ (2011) De novo discovery of mutated driver pathways in cancer. Bafna V, Sahinalp SC, eds., *Research in Computational Molecular Biology*, 499–500 (Berlin, Heidelberg: Springer Berlin Heidelberg), ISBN 978-3-642-20036-6.

Wang L, Hu S, Li M, Zhou J (2019) An exact algorithm for minimum vertex cover problem. *Mathematics* 7(7), ISSN 2227-7390, URL http://dx.doi.org/10.3390/math7070603.

Williams CP (2011) *Quantum Gates*, 51–122 (Springer London), ISBN 9781846288876, URL http://dx.doi.org/10.1007/978-1-84628-887-6_2.

Zhao J, Zhang S, Wu LY, Zhang XS (2012) Efficient methods for identifying mutated driver pathways in cancer. *Bioinformatics* 28 22:2940–7, URL https://api.semanticscholar.org/CorpusID:9075954.

Zhou Q, Xie X, Dai H, Meng W (2022) A novel rough set-based approach for minimum vertex cover of hypergraphs. *Neural Computing and Applications* 34:21793 – 21808, URL `https://api.semanticscholar.org/CorpusID:251328534`.