# Solving Travelling Salesman Problem using Quantum Computing

***Submitted in partial fulfillment of the requirements for the award of the degree of***
***Bachelor of Technology***
***In***
***Information Technology***


Submitted by
Kalyani Pharkandekar IIT2020196
Anurag Harsh IIB2020016
Aman Utkarsh IIB2020027
Priyamvada Priyadarshani IIB2020037
Dhamapurkar Nikita Suresh IIB2020042

Under the
Supervision of
Prof. Vijay Kumar Chaurasiya

**Information Technology**
**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY**
**ALLAHABAD**
**Prayagraj -211015**
**May, 2023**

# CANDIDATE DECLARATION

I hereby declare that the work presented in this report entitled "Solving Travelling Salesman Problem using Quantum Computing", submitted towards the fulfillment of BACHELOR'S MINI PROJECT report of the 6th Semester B.Tech curriculum at the Indian Institute of Information Technology Allahabad, is an authenticated record of our original work carried out under the guidance of Prof./Dr. Vijay Kumar Chaurasiya. Due acknowledgments have been made in the text to all other material used. The project was done in full compliance with the requirements and constraints of the prescribed curriculum.


Kalyani Pharkandekar
IIT2020196
Department of Information Technology

Anurag Harsh
IIB2020016
Department of Information Technology-Business Informatics

Aman Utkarsh
IIB2020027
Department of Information Technology-Business Informatics

Priyamvada Priyadarshani
IIB2020037
Department of Information Technology-Business Informatics

Dhamapurkar Nikita Suresh
IIB2020042
Department of Information Technology-Business Informatics

# CERTIFICATE FROM SUPERVISOR

This is to certify that the statement made by the candidate is correct to the best of my knowledge and belief. The project titled “Solving Travelling Salesman Problem using Quantum Computing” is a record of the candidates’ work carried out by him under my guidance and supervision. I do hereby recommend that it should be accepted in the fulfillment of the requirements of the Bachelor’s mini project at IIIT Allahabad.

Prof. Vijay Kumar Chaurasiya
Department of Information Technology

# CERTIFICATE OF APPROVAL

The forgoing thesis is hereby approved as a creditable study carried out in the area of Information Technology and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approves the thesis only for the purpose for which it is submitted.

Committee on the final examination for the evaluation of the project report:

1. Prof. Vijay Kumar Chaurasiya
   Department of Information Technology

   Signature:

1. Supervisor:
   Prof. Vijay Kumar Chaurasiya, Department of Information Technology

   Signature:

2. External Subject Expert: NA

Dean(A & R)

# Plagiarism Report

# TABLE OF CONTENTS

Supplemenatry Resources: TSP_DWAVE.ipynb - Colaboratory (google.com)

PPT:Study of TSP on Quantum Computers(C3) - Google Slides

# Abstract

Quantum computing is a rapidly emerging field with the potential to revolutionize many industries, including medicine, materials science, and finance. Quantum computers use the principles of quantum mechanics to perform calculations that are impossible for classical computers. One of the most important applications of quantum computing is solving NP-hard problems, which are problems that are difficult for classical computers to solve, even with exponential resources.
This report delves into the fascinating intersection of quantum computing and the resolution of NP-hard problems, focusing primarily on the Traveling Salesman Problem (TSP).


# I. Introduction

The Traveling Salesman Problem (TSP) stands as a renowned Combinatorial Optimization Challenge. It revolves around the task of determining the most efficient route that encompasses visiting N cities and returning to the initial city while adhering to the condition of visiting each city exactly once.

Extensively researched, the Traveling Salesman Problem is established as a member of the NP-hard complexity class. Its relevance extends across multiple domains, encompassing areas such as logistics, vehicle routing, and crystal theory. When tackled through a brute-force approach, the problem exhibits a computational complexity of (n-1)!/2, where n denotes the count of cities to be included in the journey.

Various algorithms and techniques have been since given for TSP, both exact and heuristic, techniques involving Dynamic Programming, Branch-Bound,Nearest Neighbour and Genetic algorithms are some popularly known.

These methods are generally exponential time complexity and therefore there is ever new research for alternative solutions to the problem.

Quantum Computing based on the principle of Quantum Mechanics has shown to provide significant speedups to its classical counterparts on certain problems , Combinatorial Optimization one among them.

In this paper we aim to present a study and understanding of quantum computing by applying a Quantum Optimization Algorithm on the Traveling Salesman Problem (TSP).

# II. Motivation
## A. Complexity Classes

In the realm of computational complexity theory, a complexity class can be defined as a collection of computational problems that share a common basis of resource-related complexity, as articulated by Johnson in 1990.

A complexity class therefore consists of problems that take approximately equivalent amounts of time and space to solve. Complexity classes are useful for organizing problems based on the complexity to solve and further allow us to study the relationships between these classes.

Here we provided a summary of some important classes:

**P**
The class P (Polynomial Time) consists of all decision problems (problems with either positive or negative result) that can be solved by a deterministic Turing machine in polynomial time.

**NP**
The class NP (Non-deterministic Polynomial Time) consists of all decision problems (problems with either positive or negative result) that can be solved by a non-deterministic Turing machine in polynomial time.
Or
The class NP (Non-deterministic Polynomial Time) consists of all decision problems (problems with either positive or negative result) that can be verified by a deterministic Turing machine in polynomial time.

**NP-Hard**
The class NP-hard (Non-deterministic Polynomial Time Hard) consists of all problems which every problem in NP class can be reduced to in polynomial time. Informally the NP-hard problem is as hard as the hardest problem in the NP set.

**NP-Complete**
The class NP-complete consists of problems which are both NP and NP-hard.

**TSP belongs to the NP-hard complexity set thus making it one of hard problems in optimization theory.**

## B. Motivation to use Quantum Computing

To understand the motivation of using Quantum Computing for Optimization Problems we need to understand how optimizations can be translated into a problem of a physical encoded system and eventually to a quantum system. From a fact we know that physical systems tend to be stable in their lowest energy state , referred to as the ground state. We therefore encode our information needed for optimization into the physical system, and then the goal would be to achieve the ground state.The physical state is described by a mathematical function known as Hamiltonian.

In the system we introduce constraints and whenever our constraints don't satisfy we add a penalty by increasing the Hamiltonian. In a way we are simulating different actual solutions as different states, some very close to the lowest one. Thus then we need to drive the system to the lowest state where we can apply different strategies like random search, simulated annealing or gradient descent. Simulated Annealing involves increasing the energy of a system and cooling it down gradually in order to find a minimum path, though we can stick to a local minima.

In the quantum realm there exists superpositions where each unmeasured state may be a superposition of many states , like before we drive the system to lower energy solutions but here in the quantum realm we can exploit the property of quantum tunneling which allows it to pass through higher energy states.Though even this method is prone to get stuck at local minima and hence we use quantum adiabatic algorithms, where we evolve very slowly, this allows us to get an optimal solution. This is what is the way of optimization in the quantum realm.For our problem we use quantum gate based computations using Qiskit which can simulate Hamiltonians and can perform adiabatic quantum computations. This very property of superposition and quantum tunneling gives us a new way of performing optimization which motivated us to try this.

# III. Quantum Computing

## A. The emerging field

Due to its potential to completely alter computation and problem-solving, quantum computing has grown in popularity over the past several years. An overview of the main ideas and new developments in quantum computing is given in this section.

- **Historical context -**
  Early 20th-century advances in quantum physics are where quantum computing first emerged. Significant achievements consist of:

Max Planck proposed the idea of quantization in his seminal work Planck's Quantum Theory (1900), which served as the basis for quantum physics.

Louis de Broglie's Wave-Particle Duality (1924) theory established that certain particles, including electrons, possess both wave-like and particle-like characteristics.

Schrödinger's Wave Equation (1926): Erwin Schrödinger created the wave equation, a crucial concept in quantum physics.

- **Quantum bits and modern computing challenges -**
  Qubits, the quantum equivalent of classical bits, are at the core of quantum computing. Qubits have the ability to be in several superpositions, greatly increasing their computing capacity. This computational capacity is necessary nowadays to solve complicated issues like:

  Big Data Analytics: Quantum computing offers a compelling option for the analysis of large datasets and the extraction of important insights.

  Drug Development: Quantum computing has the potential to correctly simulate molecular interactions, greatly speeding the process of drug discovery.

  Climate Modelling: Complex climate models can be simulated using quantum computers, which helps us better understand climate change and viable mitigation measures.

  Financial Modelling: Complex modelling techniques are required by contemporary financial markets, and quantum computing can provide quick risk analysis and portfolio optimisation.

- **Quantum algorithms and applications in modern world -**
  Quantum computing has the potential to make significant advances in a number of sectors, tackling problems like:

  Cybersecurity: In an increasingly digital world, it is crucial to build cryptographic algorithms that can withstand quantum attacks.

  Quantum machine learning techniques have the potential to enhance pattern identification, optimisation, and the training of AI models.

Supply Chain Optimisation: By streamlining complex supply chains, quantum algorithms can reduce costs and allocate resources more effectively.

Energy and sustainability: Quantum simulations can increase our knowledge of the components needed for sustainable development and renewable energy solutions.

## B. The Building Blocks

Essential Quantum Building Blocks: Qubits

Quantum Theory:

- Governs matter and energy behavior at the quantum level.
- Features include wave-particle duality and quantized energy levels.
- Probability amplitudes, described by complex numbers, play a crucial role.

Qubits and Their Properties:

- Quantum bits (qubits) are the core of quantum computing.
- Qubits exist in superposition, allowing parallel exploration of numerous possibilities.
- Entanglement of qubits enables strong algorithms and secure communication.
- Measurement causes qubits to collapse into defined states, extracting information.
- Qubits are represented using Dirac notation, denoted as $|0\rangle$, $|1\rangle$, or superpositions ($\alpha|0\rangle + \beta|1\rangle$).

## C. The Pre-requisites

Before we deep dive into the methodology, we need to get acquainted with the quantum mechanics terms from which originate the field of quantum computing. Rather than going deep into the proof and quantum physics full map, we will focus on the functional and useful concepts and postulates that were applied to quantum computing.

### 1. The Quantum World

“A classical computation is like a solo voice — one line of pure tones succeeding each other. A quantum computation is like a symphony — many lines of tones interfering with one another.”— Seth Lloyd

From plank to Einstein, through double slits and wave-particle duality, we've journeyed to today's quantum computing. Unlike classical bits, quantum computing uses qubits that can exist

in states of 0, 1, or both simultaneously due to superposition – a mind-bending aspect of quantum mechanics. However, this state collapses to a classical one upon observation.

Qubits can entangle, creating a cosmic connection where manipulating one qubit prompts instant reactions in its distant partner – a telepathic link across galaxies!

Adding to the intrigue, quantum interference unfolds. Picture ocean waves colliding, yielding varying effects. Quantum computers leverage this interference, processing information beyond classical dreams – akin to Jedi knights in computing, mastering intricate calculations with finesse and speed.

Yet, there's a catch. Measuring a quantum state collapses it into a definitive 0 or 1. It's like chasing a quantum butterfly – upon touch, it decides, and the outcome can be unpredictable.

Quantum computers exploit these principles to tackle tasks that classical computers struggle with, from breaking encryption codes to simulating molecules for drug discovery and optimizing supply chains with unparalleled efficiency.

However, the plot twist is building a quantum computer isn't easy. These machines are fragile, sensitive to their surroundings, and demand supercooling to near absolute zero. Despite the challenges, scientists and engineers are on a mission, racing to democratize quantum computing. The potential is vast, and the quantum future promises a mind-bending adventure.

## 2. Quantum States & Hilbert space

In the quantum realm, envision the quantum state as a mystical recipe revealing everything about a particle's existence – a complex narrative far beyond everyday descriptions. Quantum states, residing in Hilbert space, are intricate and hold vast information about a particle, encompassing aspects like spin and polarization.

Hilbert space, a mathematical playground for quantum states, is vital. Unlike classical coordinates, it deals with abstract, multi-dimensional spaces, each dimension capturing a facet of the particle's state. This becomes crucial when handling quantum systems with many particles, such as atoms or molecules, where Hilbert space elegantly describes and manipulates a myriad of possible states.

In the quantum dance of possibilities, Hilbert space is the stage where particles unveil their secrets. It's the mathematical tool essential for exploring the mysteries of the quantum universe and advancing quantum physics and computing.

Key Characteristics of Hilbert Space:

Vector Space: Elements can be linearly combined.
Inner Product: Allows angle and distance calculations between vectors.
Complex Numbers: Vectors can have complex components, enabling quantum state descriptions.
Complete: Every Cauchy sequence in the space converges within the space.
Infinite Dimension: Accommodates quantum systems with diverse properties.
Separable: Contains a countable dense subset, facilitating mathematical analysis in quantum mechanics.
Unitary Transformations: Quantum operations are often represented by unitary transformations in Hilbert space, preserving the inner product and norm.

## 3. Bra-Ket notation

Bra-ket notation serves as the efficient and compact language of quantum mechanics, akin to a secret code for describing quantum states and operations. The 'bra' ($\langle\psi|$) represents the complex conjugate of a quantum state, serving as the starting point, while the 'ket' ($|\varphi\rangle$) signifies the quantum state itself, akin to the desired endpoint.

When combined ($\langle\psi|\varphi\rangle$), this notation acts as a powerful handshake, conveying information about the likelihood of measuring the state $|\varphi\rangle$ when starting with $|\psi\rangle$ or how a quantum operator transforms $|\psi\rangle$ into $|\varphi\rangle$.

Moreover, the bra notation for a state is the conjugate transpose of the ket, forming a quantum ninja move that enables scientists to elegantly and precisely describe quantum phenomena. The 'bra' and 'ket' symbols offer a glimpse into the secret language of the quantum realm, making complex concepts more manageable.

Notably, while Bra-ket provides a form similar to the scalar/dot product (inner product), ket-bra yields the density matrix, crucial for understanding decoherence properties in quantum computing. Decoherence, involving the loss or degradation of quantum information due to interactions with the environment, is a fundamental challenge in building practical and robust quantum computers.

$$|\psi\rangle \doteq \begin{pmatrix} a_\psi \\ b_\psi \end{pmatrix}$$

Fig : Representation of a quantum state Bra–ket notation - Wikipedia

$$|\psi\rangle^\dagger = \langle\psi|$$

Fig :Conjugate transpose of each other [Bra–ket notation - Wikipedia]

## 4. Unitary Operations

Unitary operations, the superheroes of quantum vector space, preserve the 'quantumness' of states. In this space, quantum states are represented as evolving vectors, conserving probabilities and maintaining unique properties. Unitary operations, akin to wizards, apply a magical twist, changing the state's direction without losing quantum properties. Mathematically, unitary operations, represented by special matrices, ensure preservation of probabilities, inner products, and quantum attributes. They act as guardians, allowing quantum states to dance and evolve while retaining their charm, making computations and transformations elegant and reliable. The operation $U|\psi\rangle=|\varphi\rangle$ symbolizes a norm-preserving and reversible transformation.

## 5. Eigenvectors & Eigenvalues

Eigenvector: In linear algebra, an eigenvector for a square matrix A is a non-zero vector that, when multiplied by A, results in a rescaled version of itself ($Av = \lambda v$). These vectors symbolize directions in the vector space that undergo stretching or compression during the matrix A's linear transformation, holding significance in applications like identifying principal components in data analysis.

Eigenvalue: An eigenvalue is a scalar linked to a specific eigenvector. In the equation $Av = \lambda v$, $\lambda$ represents the eigenvalue, providing insight into the extent of stretching or compression experienced by the eigenvector when matrix A is applied. Eigenvalues play a paramount role in understanding linear transformation characteristics and find applications in various mathematical and scientific domains.

Eigensolver: A computational algorithm like the power iteration, QR algorithm, or Lanczos algorithm is used to find the eigenvalues and corresponding eigenvectors of a matrix. Since analytical solutions for large matrices are challenging, these numerical methods efficiently approximate eigenvalues and eigenvectors.

## 6. Schrödinger Equation & Hamiltonian

**Schrödinger Equation:**

In the quantum realm, particles are described by mysterious wavefunctions – complex-valued functions of space and time, acting as vectors in a complex vector space. The Schrödinger equation is the conductor's baton, a partial differential equation guiding the evolution of these wavefunctions over time, revealing how quantum states change.

Magic unfolds in its solutions, akin to golden tickets, unveiling energy levels of the quantum system. Each solution corresponds to a specific allowed energy state, precisely describing the associated wavefunctions.

**Hamiltonian Operator:**
The Hamiltonian operator (H) is the quantum treasure map representing the total energy of the system. As a differential operator, it acts on the wavefunction to calculate the system's energy. Solving the time-independent Schrödinger equation reveals eigenvectors and eigenvalues of the Hamiltonian operator, unlocking the quantum system's energy states.

In essence, the Schrödinger equation and Hamiltonian operator are quantum storytellers, narrating wavefunction evolution and unveiling hidden energy levels, fundamental in understanding and predicting quantum system behavior.

**Time-dependent**

$$i\hbar\frac{d}{dt}|\Psi(t)\rangle = \hat{H}|\Psi(t)\rangle$$

Fig: Time-dependent equation Schrödinger equation - Wikipedia

The time-dependent Schrödinger equation, $i\hbar\partial\Psi/\partial t = H\Psi$, describes how the wavefunction $\Psi$ of a quantum system changes over time. It connects the system's energy ($H\Psi$) to its evolving quantum state ($\partial\Psi/\partial t$), serving as a foundational equation in quantum mechanics. Solving it allows prediction and understanding of quantum system behavior, making it a cornerstone in quantum mechanics.

## 7. Classical Ising Hamiltonian

Imagine a cast of tiny magnets, each with its unique personality based on atomic arrangements. Some prefer north, some south, and others have special angles they fancy. Bringing them close initiates magnetic conversations – a cuddling phenomenon called "coupling." The Ising model acts as a script for this magnetic play, helping understand how magnets, each with biases and interacting with neighbors, collectively behave – forming groups or scattering.

Introduced biases using an external magnetic field add preference. The classical Ising Hamiltonian, named after Ernst Ising, describes the behavior of magnetic spins or binary variables. For a set of binary variables $\sigma i$ (like magnets), the Ising model's energy equation is

$E_{total} = -\sum(h_i * \sigma_i) - J * \sum(\sigma_i * \sigma_j)$, where the first summation is over all magnets and the second is over neighboring pairs. The goal is to find the σi configuration minimizing the energy of the system, optimizing H(σ).

## 8. Transverse field

So far, we've been considering magnets that naturally prefer to align with an external magnetic field (the bias). Now, imagine we have another influence on our magnets – a field that acts perpendicular to their natural alignment, like a gentle breeze nudging them sideways.
The transverse field is like an external force that can flip a magnet's orientation, regardless of its natural bias. We represent this influence using another parameter, denoted as Γ.

Now, let's add the transverse field term to our Ising model's energy equation:

$E_{total} = -\sum(h_i * \sigma_i) - J * \sum(\sigma_i * \sigma_j) - \Gamma * \sum\sigma_i$

Here, the third term, $-\Gamma * \sum\sigma_i$, represents the influence of the transverse field. It sums up the contributions of the transverse field acting on each individual magnet $\sigma_i$.

The presence of the transverse field can lead to fascinating behaviors in the Ising model. It can induce quantum effects and phase transitions, making our magnetic ensemble dance in unexpected ways. Indeed, finding the optimal value of Γ is often a critical part of studying the Ising model with a transverse field. It depends on various factors, including the strength of the natural biases ($h_i$) and the coupling strength (J) between magnets. Making it a Np-hard problem, impossible to be solved by classical computers.

## 9. Conversion of Classical Ising to Quantum realm

So, our magnetic friends were quite set in their ways in the classical world, always pointing either up or down, right? But now, let's introduce them to a quantum realm where they can do something truly enchanting: exist in multiple states at once!
Instead of just having two choices (+1 or -1), our quantum magnets can exist in a superposition of states. It's as if they can simultaneously be a bit up and a bit down, and everything in between. We represent this magical property using quantum spin operators, like $\sigma_x$, $\sigma_y$, and $\sigma_z$.

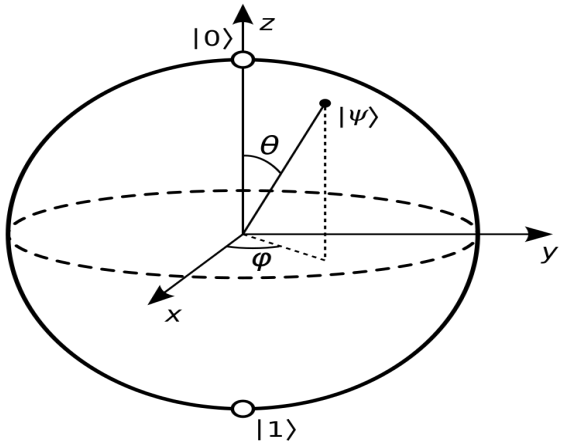



Fig: Bloch sphere (visual representation of qubits which are in actuality wave functions)
Quantum logic gate - Wikipedia

We keep the bias, represented by h_i, but now it becomes a quantum bias. The quantum magnets can still have a preference to point up or down along the z-axis.

For the coupling between neighboring magnets, we use quantum operators as well. The interaction term is similar to the classical one, but now it's described using quantum spin operators, like σ_z□ and σ_z□'.

Transverse field acts on our quantum magnets, but instead of pushing them up or down, it nudges them sideways, along the x-axis. This field can make our quantum magnets flip and dance between different states. In the quantum Ising model, our magnets become like graceful ballet dancers, twirling and spinning through various quantum states, thanks to the transverse field's enchantment.

In terms of mathematics, we replace the classical spins with quantum spin operators and adjust the Ising Hamiltonian accordingly. We use operators like σ_x, σ_y, and σ_z to describe the quantum spins, and the Hamiltonian includes terms for bias, coupling, and the transverse field, just like in the classical model.

## 10.Quantum Complexity theory

Quantum Complexity:
Quantum complexity theory focuses on specialized quantum algorithms designed for quantum computers, aiming for more efficient problem-solving than classical algorithms. Notable algorithms include Shor's for integer factorization and Grover's for unstructured search.

Quantum Classes:

In quantum complexity, we introduce classes like BQP (bounded-error quantum polynomial time), analogous to classical P, defining problems solvable by quantum computers in polynomial time with a small error probability.

Oracle Queries:
Quantum complexity explores oracle queries, representing instant problem-solving black boxes. Quantum oracles play a crucial role in analyzing the efficiency of quantum algorithms.

Hierarchies:
Quantum complexity introduces hierarchies like the polynomial-time hierarchy in quantum computing (QPH), classifying the complexity of problems solvable by quantum computers.

Speedup and Limitations:
Identifying problems with substantial quantum speedup is a central question. While quantum computers excel in specific domains, understanding their limitations and advantages is crucial.

Conjectures:
Quantum complexity theory, akin to classical complexity, has conjectures. Notable is the Quantum Unique Games Conjecture (QUGC), influencing the hardness of quantum optimization problems.

Entanglement and Beyond:
Entanglement enables remarkable results in quantum algorithms, connecting to quantum information theory. Quantum complexity provides a rigorous framework, bridging mathematical rigor and the potential for groundbreaking advancements in computation, essential to the future of quantum physics and computing.

## 11. Quantum Approximate Optimization Algorithm (QAOA)

The Quantum Approximate Optimization Algorithm (QAOA) holds a prominent position in the realm of quantum computing, particularly when it comes to addressing combinatorial optimization challenges on near-term quantum devices referred to as NISQ (Noisy Intermediate-Scale Quantum) devices. QAOA is noted for its broad practical utility and has garnered substantial attention within the quantum computing research community due to its potential to address real-world problems effectively. The QAOA process typically unfolds in iterative fashion. During each cycle, we make gradual adjustments to the circuit's parameters, incrementally refining our solution. It's akin to ascending a mountain peak, progressively approaching the optimal path with each step.

We can the approximation of the time evolution of a quantum system operator as:

$$U(t, t_0) = U(t_n, t_{n-1})U(t_{n-1}, t_{n-2})...U(t_2, t_1), U(t_1, t_0)$$

Fig: Quantum Approximate Optimization Algorithm Explained | by Thomas Lawrence | Medium

where n is the number of "points". So the bigger the n the more accurate the approximation is.

$$|\gamma, \beta\rangle = U(H_B, \beta_p)U(H_c, \gamma_p)...U(H_B, \beta_1)U(H_c, \gamma_1)|s\rangle$$

Fig: Quantum Approximate Optimization Algorithm Explained | by Thomas Lawrence | Medium

Additionally, "trotterization" is a technique frequently employed in quantum algorithms, particularly in the context of quantum simulation, where it's used to approximate the time evolution of quantum systems, making it a valuable tool in various quantum computations.

Variational circuits are the intuition applied here according to which you parametrize the quantum circuit after running short best and fine-tuning repetitively.

## 12.Quadratic unconstrained Binary Optimization (QUBO )

Short for Quadratic Unconstrained Binary Optimization, and it's a math problem that goes like this: In QUBO, the idea is to pick binary values, which means either 0 or 1, for a bunch of choices you have. Each of these choices can be thought of as a simple yes (1) or no (0). Your job is to figure out the best combo of choices that gives you the lowest total cost. This cost is calculated using a fancy math equation that involves these choices.

Mathematically, a QUBO problem can be written as:
minimize: f(x) = ∑(i, j) Qij * xi * xj
subject to: xi ∈ {0, 1} for all i
In this equation:
xi represents binary decision variables.
Qij represents the coefficients that determine the cost or benefit associated with selecting combinations of variables xi and xj.

The objective is to find the values of xi (0 or 1) that minimize the total cost f(x).
QUBO problems find applications in various fields, including optimization, machine learning, and quantum computing. They serve as a fundamental framework for solving discrete optimization problems by mapping them onto this mathematical model. The challenge lies in

determining the appropriate coefficients Qij to encode a real-world problem into the QUBO form, and then employing optimization techniques to find the optimal binary solution.
Six simple inferences we drew in general:

1. The wave function determines everything
2. All observables have operators
3. Observables are hermitian
4. Eigenfunctions of operators are independent
5. Expectation value integral
6. Time-dependent Schrödinger equation

# D. The Models

Practically, all classical computers work in a similar way. A bunch of bits holding the binary information of 1 and 0. We can perform computations on these bits using logical gates. However, Quantum computing has several models. Discussing the most useful and famous models of quantum computing used all over the world.

## 1. Gate based Models

Universal way of transforming quantum states into another quantum state.(Performing computations)

### a) Qubits and Gates

Quantum gates are like the essential LEGO pieces in quantum algorithms. Just as classical logic gates manipulate bits, these gates are operators that work their magic on qubits. But here's where it gets interesting: unlike their classical counterparts, quantum gates can tap into superposition and entanglement to perform intricate tasks. Among the noteworthy quantum gates, you've got the Hadamard gate and the CNOT gate. What's really cool is that they can perform their tricks on qubits without checking their values until the very end.

When qubits become entangled, their states become inherently correlated, irrespective of spatial separation. This property enables quantum computers to perform tasks that are classically intractable, such as factorization or quantum teleportation.

Gates applied to entangled qubits in an order is a quantum algorithm to change the probability of the final state of a qubit when it is finally measured.

Solovay-Kitaev theorem:
We can approximate any quantum gate using a specific set of basic gates. The Solovay-Kitaev theorem assures us that we can achieve universality – the ability to approximate any quantum operation – using just a finite set of gates. It helps us design error-correcting codes and

fault-tolerant quantum algorithms by approximating our noisy gates with sequences of more reliable basic gates.

## b) Types of Gates

Let's meet some of the main types:

1. Pauli-X Gate (σ_x):

Professor: The Pauli-X gate is like a quantum flipper. It changes the state of a qubit from |0⟩ to |1⟩ and vice versa. Imagine it as a magical mirror that reflects the quantum state across the equator of the Bloch sphere.

2. Pauli-Y Gate (σ_y):

Professor: The Pauli-Y gate is our quantum twister. It rotates the qubit state around the y-axis. It's as if our qubit is performing an elegant ballet spin in the quantum realm.

3. Pauli-Z Gate (σ_z):

Professor: The Pauli-Z gate is like a quantum compass. It doesn't change the qubit state but adds a phase. It's as if our qubit is rotating around the north and south poles of the Bloch sphere.

4. Hadamard Gate (H):

Professor: The Hadamard gate is a quantum magician. It transforms |0⟩ to a superposition state, allowing the qubit to exist in both |0⟩ and |1⟩ simultaneously. It's like splitting a quantum coin into a quantum coin toss.

5. CNOT Gate (Controlled-X):

Professor: The CNOT gate is a quantum communicator. It operates on two qubits: one acts as a control and the other as a target. Depending on the control qubit's state, it flips the target qubit's state. Think of it as quantum telepathy between qubits.

6. T Gate (π/4 Phase Gate):

Professor: The T gate is our quantum clock hand. It adds a π/4 phase to the qubit state. It's like gently nudging the quantum state clockwise around the Bloch sphere.

7. SWAP Gate:

Professor: The SWAP gate is a quantum rearranger. It swaps the states of two qubits, allowing them to exchange places. Imagine it as a pair of qubits doing a graceful dance.

8. Ry Gate (Rotation around the y-axis):

Professor: The Ry gate is a quantum spinner. It rotates the qubit state around the y-axis by a specified angle, allowing precise control of the qubit's orientation.

9. CZ Gate (Controlled-Z):

Professor: The CZ gate is a quantum gatekeeper. Similar to the CNOT gate, it operates on two qubits but adds a phase to the target qubit if the control qubit is in the |1⟩ state. It's like placing a quantum lock on the gate.

## c) Example

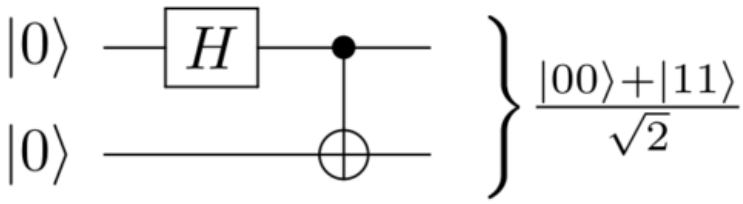



Fig: The Hadamard-CNOT gate, which when given the input |00⟩ Quantum logic gate - Wikipedia

Solution:

$$\mathrm{CNOT}(H \otimes I)|00\rangle = \left( \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \left( \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \right) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Fig: Quantum logic gate - Wikipedia

## d) Software Process

Slightly different from what we are used to in classical computers, One must first define what problem and find a matching quantum algorithm like QAOA, which will split into gates and

unitary operations finally. Below this level, the actual compilation is going on. If the set of gates they are not implemented by a quantum computer it does a translation. It considers actual actual set of gates implemented in the hardware. For interaction between two qubits that are not physically connected, they also deal with connectivity. Finally, execute on system/simulator.

## 2. Adiabatic Quantum Computing

It can also transform qs to another qs. To understand how it can achieve universal computation, we have to understand more of the underlying physics
Hamiltonian describes the energy of the system and unitary that describes the evolution of the system.

Energy expectation value = $\langle H \rangle = \langle \psi | H | \psi \rangle$
The expression $\langle \psi(\theta) | H | \psi(\theta) \rangle$ represents the expectation value of an operator H in a quantum state $|\psi(\theta)\rangle$ parameterized by the set of parameters $\theta$.

And the evolution is $U|\psi\rangle$
Both are of exact same form the correspondence is described by the Schrodinger equation. The temporal evolution of the system is described by the Hamiltonian applied to the time-dependent state. I is imaginary and h is plank constant.
Time-independent
When we solve in case Hamiltonian does not depend on time we get this formula which is exactly the unitary acting on a particular state for a duration t.

$$f(t) = e^{\frac{iEt}{\hbar}}$$

Fig: Time-independent Schrodinger equation Schrodinger's Equation: Explained & How to Use It | Sciencing

So, every unitary/gate has an underlying Hamiltonian. Hamiltonian is always a hermitian operator which means its adjoint is itself which implies why this operation is unitary.

Adiabatic quantum computation:
Imagine you have two Hamiltonian, one tunneling Hamiltonian or initial Hamiltonian or age zero or just transverse field acting on a number of sites and the ground state is equal superposition. The other Hamiltonian is problem Hamiltonian.

H(t) = (1-t)H0 + tH1 (Time-dependent mixing the two) t = [0,1]
H1 represents the classical Ising model within this context. When we make a slow enough change over time while beginning from the lowest energy state of H0, we ultimately arrive at the

lowest energy state of H1, solving the problem along the way. Nature faces a challenging task in accomplishing this because the system can sometimes get stuck in a local best solution. However, we employ an adiabatic transition here, ensuring a smooth progression from the ground state and maintaining a lower energy state throughout the transformation. This allows us to extract the global solution to our problem. However, there's a catch: there exists a speed limit inversely tied to the square of the minimum gap.

AQC finally is given by Hamiltonian of classical Ising model + term which is the interaction between transverse field (more than just the effect the transverse fields alone)

## 3. Quantum Annealing

Less idealized version of AQC. Performs AQC again and again and to reach the solution.
The physics of this phenomenon can be effectively represented through an energy diagram, as illustrated in Figure below. This diagram undergoes dynamic changes over time, as depicted in (a), (b), and (c). Initially, there exists a single valley (a) characterized by a solitary minimum point. As the quantum annealing process unfolds, a barrier is progressively elevated, transforming the energy diagram into what is referred to as a "double-well potential" (b). In this configuration, the lower point within the left valley corresponds to the 0 state, while the lower point within the right valley corresponds to the 1 state. At the conclusion of the annealing process, the qubit ultimately settles within one of these valleys.

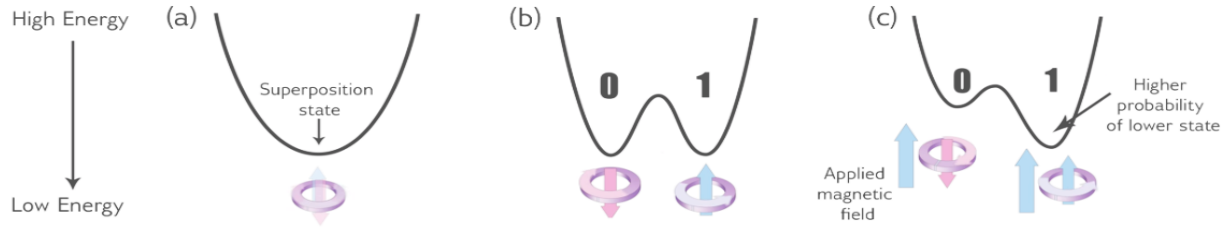



Fig: [Getting Started with D-Wave Solvers — D-Wave System Documentation documentation (dwavesys.com)]

Quantum systems, exhibiting wave-particle duality, can escape local optima by tunneling through barriers.

In quantum computing, qubits' probabilities are controlled by programmable biases known as "bias," and their interaction is facilitated by "couplers," leveraging the entanglement phenomenon. This entanglement connects qubits, allowing them to influence each other, enhancing computational capabilities.

The collective use of programmable biases and couplers defines the problem on a D-Wave quantum computer. Entanglement occurs when qubits become linked, behaving as a single unit with four possible states. The annealing process involves navigating the energy landscape influenced by biases and coupling strength, eventually converging into a specific configuration, such as (1,1), as the annealing concludes.

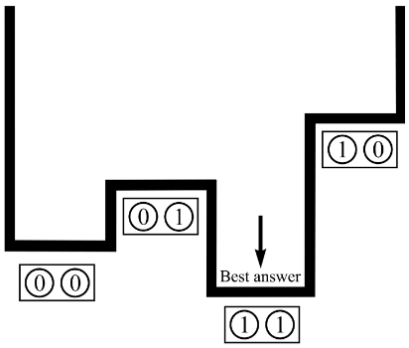



Fig:Getting Started with D-Wave Solvers — D-Wave System Documentation documentation (dwavesys.com)

Biases and couplings sculpt an energy landscape, and D-Wave quantum computers venture to uncover its minimal energy state, a process known as quantum annealing.

$$\mathcal{H}_{ising} = \underbrace{-\frac{A(s)}{2}\left(\sum_i \hat{\sigma}_x^{(i)}\right)}_{\text{Initial Hamiltonian}} + \underbrace{\frac{B(s)}{2}\left(\sum_i h_i \hat{\sigma}_z^{(i)} + \sum_{i>j} J_{i,j}\hat{\sigma}_z^{(i)}\hat{\sigma}_z^{(j)}\right)}_{\text{Final Hamiltonian}}$$

Fig:Getting Started with D-Wave Solvers — D-Wave System Documentation documentation (dwavesys.com)

The Complexity of the system increases exponentially with the addition of ore qubits 4 state for 2 qubits, 8 for 3 likewise.

Energy Landscpae/EigenSpectrum:

In the realm of quantum systems, a Hamiltonian serves as a guiding map, linking specific states known as eigenstates to their corresponding energies. It's only when the system precisely aligns with an eigenstate of the Hamiltonian that its energy assumes a clear and distinct value, termed the eigenenergy. In contrast, when the system resides in any different state, the energy remains uncertain, much like the ever-shifting sands of possibility. This ensemble of eigenstates, each with its well-defined eigenenergy, collectively constitutes what we call the "eigenspectrum."

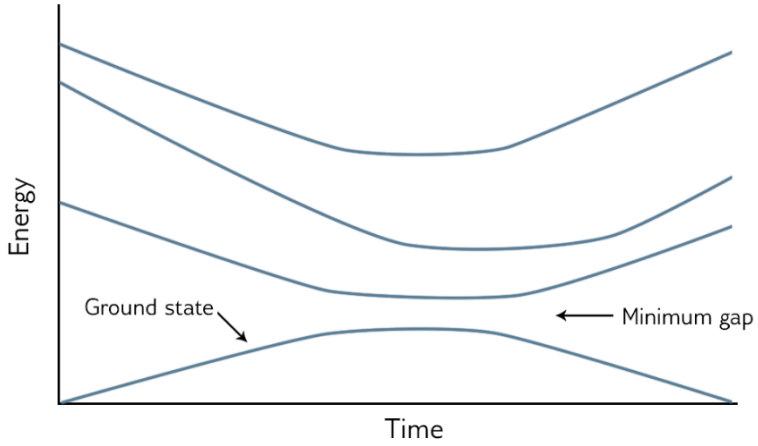



Fig: [Getting Started with D-Wave Solvers — D-Wave System Documentation documentation (dwavesys.com)]

In quantum annealing, the process begins with the quantum system in its lowest energy state, distinct from other levels. As the problem Hamiltonian is introduced, certain energy levels approach the ground state. The minimum gap, the smallest separation between the ground state and the nearest excited state during the anneal, is a crucial measure.

Factors like thermal fluctuations or rapid annealing execution can induce transitions to higher energy states. Achieving an adiabatic process, where the Hamiltonian evolves gradually without external disruptions, is theoretically ideal but challenging in real-world applications.

Quantum annealing, a practical counterpart to adiabatic quantum computing, identifies low-energy states valuable for problem-solving, even if staying in the ground state is improbable. The difficulty of quantum annealing is often tied to the minimum gap, making problems with small minimum gaps more challenging.

## E. Comparison

D-Wave has successfully implemented quantum annealing in their Advantage™ model, a specialized approach for solving optimization problems. Quantum annealing utilizes intrinsic quantum effects to address specific problems, framing them as energy minimization challenges. This method is scalable, enabling the construction of quantum processing units (QPUs) with a substantial number of qubits.

In quantum annealing, the process involves setting up the initial state, providing an environment, and letting quantum physics naturally evolve to find solutions. It's effective for optimization problems and sampling challenges, where the goal is to characterize the energy landscape probabilistically.

While quantum annealing is not a universal quantum computer, it has been scaled to over 5000 qubits. In contrast, the gate-based model, aiming for greater control over quantum state evolution, faces challenges due to the delicacy of quantum systems. Presently, only around 1000 qubits exist for the gate-based model. Quantum annealing is related to adiabatic quantum computing, which is a universal model, but the gate-based model's difficulty in realization limits its scalability.

## F. Latest development

Recent Advances in Quantum Computing:

Quantum Hardware Advancements:

- Trend towards more qubits and improved qubit quality.
- Quantum volume metric assesses processing power considering qubit count and error rates.

Quantum Supremacy:
Google claimed quantum supremacy in 2019, demonstrating computing tasks beyond traditional supercomputers' capabilities.

Quantum Cloud Services:

- Emergence of cloud-based quantum computing services from IBM, Amazon, and Microsoft.
- Provides easier access for researchers and developers to test quantum algorithms.

Quantum Software Ecosystem:
Rapid advancement of quantum programming languages and software libraries.
Examples include Qiskit, Cirq, and Forest for writing and running quantum algorithms.

Quantum Error Correction:

- Intensive study of quantum error correcting codes to mitigate faults in quantum calculations.
- Essential for scaling quantum systems to solve complex problems consistently.

Quantum Machine Learning:

- Exploration of the potential superiority of quantum machine learning algorithms over traditional methods.
- Investigating algorithms like quantum support vector machines and quantum neural networks.

Quantum Cryptography Advancements:
Development of quantum-safe cryptographic protocols for secure communication in a post-quantum era.

Quantum Simulations:
Increased use of quantum computers to simulate quantum systems in materials science, chemistry, and complex physical processes.

Quantum Annealers:
Constant development of quantum annealing platforms, like those from D-Wave Systems, for optimization issues and job sampling.

Quantum Education and Research:
Expansion of educational initiatives and research efforts to train the next generation of quantum scientists and engineers.

Quantum Ethics and Policy:

- Growing discussions on ethical, security, and policy implications of advancing quantum technologies.
- Consideration of the impact on national security and privacy by policymakers.

## IV. Tools, Framework, and Technologies used

1. ***Qiskit:*** Qiskit is an open-source software development kit for working with quantum computers at the level of circuits, algorithms, and application modules! This SDK allows anyone to program on real quantum computers from their laptop. It enables users to design and manipulate quantum circuits using a Python-based interface. It offers quantum simulators for testing algorithms and access to IBM's quantum devices through the cloud. The framework includes implementations of various quantum algorithms and tools for quantum-enhanced optimization and machine learning. Qiskit has a thriving community, extensive documentation, and educational resources, making quantum computing accessible to a wide range of users.

2. ***IBM Quantum***: IBM Quantum is a leading provider of quantum computing hardware, software, and cloud services. IBM offers a variety of quantum processors, from small devices with a few qubits to larger devices with hundreds of qubits. It also offers a suite of quantum software tools and libraries, as well as cloud access to its quantum processors. It is working with researchers, developers, and businesses all over the world

to explore the potential of quantum computing to solve real-world problems and is also committed to making quantum computing more accessible and affordable for everyone. In short, IBM Quantum is a major player in the field of quantum computing, and it is working to make this technology a reality for everyone.

3. ***Python***: Programming language used to implement the quantum computing algorithms.

4. ***Jupyter Notebook***: The Jupyter Notebook is a web-based application for creating and sharing computational documents. It is a popular tool for data science, scientific computing, and machine learning because of its simplicity and flexibility.

# V. Proposed Methodology

The Travelling Salesman Problem (TSP) is a well-known combinatorial optimization problem in the field of computer science, operations research, and mathematics. It is often used as a benchmark problem for optimization algorithms and has a wide range of real-world applications, including logistics, transportation, manufacturing, and circuit design.

In the Traveling Salesman Problem, we are given a set of cities, and the goal is to find the shortest possible route that visits each city exactly once and returns to the starting city. The problem can be formally defined as follows:

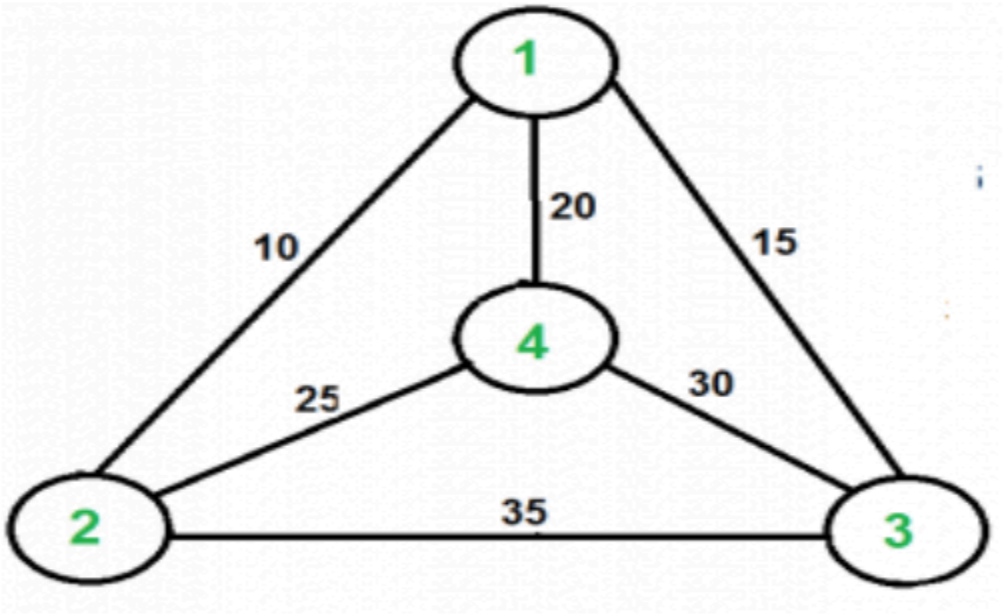
1. Given a set of N cities {C1, C2, ..., Cn}.
2. Find a permutation (ordering) of these cities, such as (C1, C2, ..., Cn), that minimizes the total distance traveled when visiting each city exactly once and returning to the starting city.

The total distance traveled is usually defined as the sum of the distances between consecutive cities in the permutation. The distances between cities are typically represented by a distance matrix, where the entry in the i-th row and j-th column represents the distance between city $C_i$ and city $C_j$.

The Traveling Salesman Problem is classified as an NP-hard problem, which means that as the number of cities (N) increases, the number of possible permutations grows factorially (N!), Therefore, various algorithms and heuristics have been developed to find approximate solutions or good-quality solutions in a reasonable amount of time. Some of the popular algorithms for solving the TSP include - Brute Force and Dynamic Programming approaches.

## TSP as a graph problem

The Traveling Salesman Problem (TSP) can be conceptualized as an undirected graph where cities represent the graph's nodes, the routes between cities are the graph's edges, and the length of a route corresponds to the weight of an edge. This problem involves finding the shortest possible route that starts and ends at a predetermined city while visiting all other cities exactly once. In many instances, this model assumes a complete graph, meaning that there is a direct route (edge) between every pair of cities. To illustrate this concept, let's examine the following example graph -




A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80.

**Note -** The distinction between the Hamiltonian cycle problem and the Traveling Salesman Problem (TSP) lies in their specific goals. The Hamiltonian cycle problem is concerned with determining whether a tour exists that covers each city precisely once. In contrast, the TSP assumes that a Hamiltonian tour is guaranteed to exist (since the graph is complete, connecting all cities), and the primary objective is to identify the Hamiltonian Cycle with the least total weight among the multiple possible tours.

# Solving TSP on Classical Computers - Brute Force Approach

## Draw a graph:

Defines a function called *draw_graph*. The function takes three arguments:
G: A NetworkX graph object
colors: A list of colors, one for each node in the graph
pos: A dictionary of node positions

Create new axes object and set the *frameon* parameter to *True*. This means that the axes will be visible when the graph is drawn.

Draw the graph on the axes object. The *node_color* parameter specifies the color of each node, the *node_size* parameter specifies the size of each node, the *alpha* parameter specifies the transparency of each node, the *ax* parameter specifies the axes object to draw the graph on, and the *pos* parameter specifies the position of each node.

Get the edge labels for the graph. The edge labels are stored in a dictionary, where the key is the edge tuple and the value is the weight of the edge.

Draw the edge labels on the graph. The *pos* parameter specifies the position of each node, and the *edge_labels* parameter specifies the edge labels.

```
[7]: def draw_graph(G, colors, pos):
         default_axes = plt.axes(frameon=True)
         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")
         nx.draw_networkx_edge_labels(G, pos=pos, edge_labels=edge_labels)
```

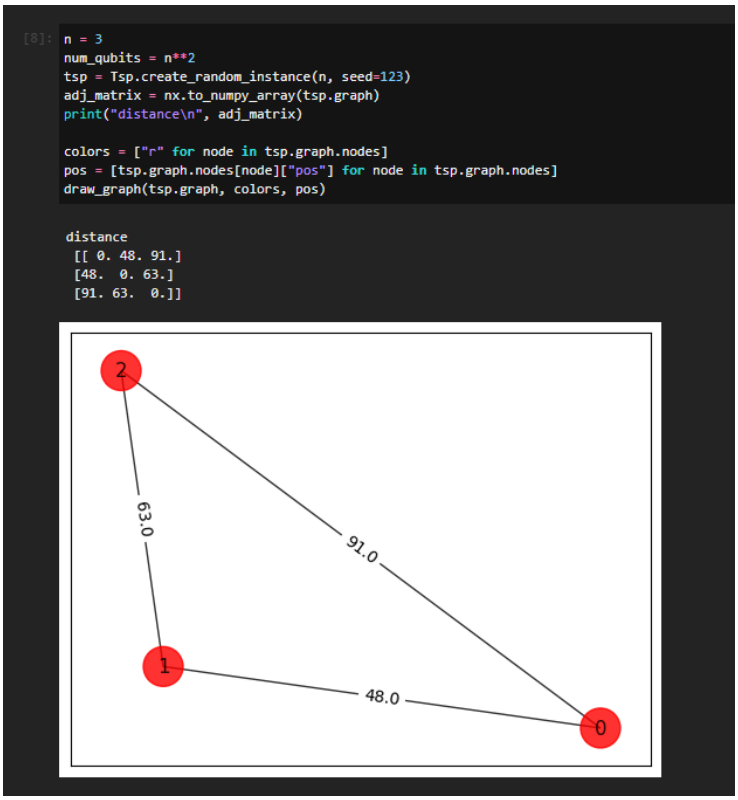
Define the number of qubits and create a random TSP instance:
We are setting the number of qubits (n) to 3, and then calculating the total number of qubits (*num_qubits*) needed to represent the TSP problem using a quantum annealer. Next, you're creating a random TSP instance using the *Tsp.create_random_instance()* function. The *seed* parameter ensures reproducible results for the random instance generation.

Convert the TSP graph to an adjacency matrix:
The *adj_matrix* represents the distances between all pairs of nodes in the TSP graph. It's obtained by converting the NetworkX graph object (*tsp.graph*) to a NumPy array using the *nx.to_numpy_array()* function.

Print the distance matrix:
Prints the distance matrix to the console, allowing you to visualize the distances between all nodes.

Set node colors and positions:
Create a list of colors (*colors*) where each node is represented by the color red ("*r*").
Additionally, you're creating a list of node positions (*pos*) by extracting the *pos* attribute for each node from the TSP graph (*tsp.graph.nodes*).

Draw the TSP graph:
Call the *draw_graph()* function to visualize the TSP graph on a canvas. The function takes the TSP graph (*tsp.graph*), the list of node colors (*colors*), and the list of node positions (*pos*) as arguments.




## The brute force approach

Solving the Traveling Salesman Problem (TSP) using classical computers through a brute-force approach involves systematically examining all possible permutations of cities to find the shortest possible tour. The steps are as follows -

1. Begin by selecting any city as both the starting and ending point of the tour since the route forms a closed loop, and the choice of the initial city doesn't affect the solution.
2. Next, create all possible permutations of cities except for the starting city. For example, if there are N cities, generate (N-1)! permutations. These permutations represent different orders in which the cities can be visited.
3. For each generated permutation, compute the total cost or distance of the tour by summing up the distances between consecutive cities as dictated by the order in the permutation. Keep track of the permutation with the lowest total cost encountered so far.
4. Finally, return the permutation that corresponds to the minimum cost found. This permutation represents the optimal solution to the Traveling Salesman Problem, where the tour begins and ends at the chosen city, and all other cities are visited exactly once along the way.

[10]:
```
def brute_force_tsp(w, N):
    a = list(permutations(range(1, N)))
    last_best_distance = 1e10
    for i in a:
        distance = 0
        pre_j = 0
        for j in i:
            distance = distance + w[j, pre_j]
            pre_j = j
        distance = distance + w[pre_j, 0]
        order = (0,) + i
        if distance < last_best_distance:
            best_order = order
            last_best_distance = distance
            print("order = " + str(order) + " Distance = " + str(distance))
    return last_best_distance, best_order
```

[11]:
```
best_distance, best_order = brute_force_tsp(adj_matrix, n)
print(
    "Best order from brute force = "
    + str(best_order)
    + " with total distance = "
    + str(best_distance)
)
```

```
order = (0, 1, 2) Distance = 202.0
Best order from brute force = (0, 1, 2) with total distance = 202.0
```

## Draw TSP Graph

Define a function called *draw_tsp_solution* that takes four arguments: *G, order, colors,* and *pos*. *G* is the NetworkX graph object representing the TSP problem, *order* is the list of cities in the optimal solution, *colors* is the list of node colors, and pos is the list of node positions.

Create a directed graph object (*G2*) using the NetworkX library.

Add all the nodes from the original graph (G) to the directed graph (G2).

Calculate the length (n) of the list of cities (order).

Start a for loop that iterates over the list of cities (order), where i represents the index of the current city.

Calculate the index (j) of the next city in the list, considering the circular nature of the TSP problem.

Add an edge to the directed graph (G2) between the current city (order[i]) and the next city (order[j]). The weight of the edge is set to the distance between the two cities, which is obtained from the original graph (G).

Create a new axes object (default_axes) and set the *frameon* parameter to True. This means that the axes will be visible when the graph is drawn.

Draw the directed graph (G2) on the axes object. The *node_color* parameter specifies the color of each node, the *edge_color* parameter specifies the color of the edges, the *node_size* parameter specifies the size of each node, the *alpha* parameter specifies the transparency of each node, the *ax* parameter specifies the axes object to draw the graph on, and the pos parameter specifies the position of each node.

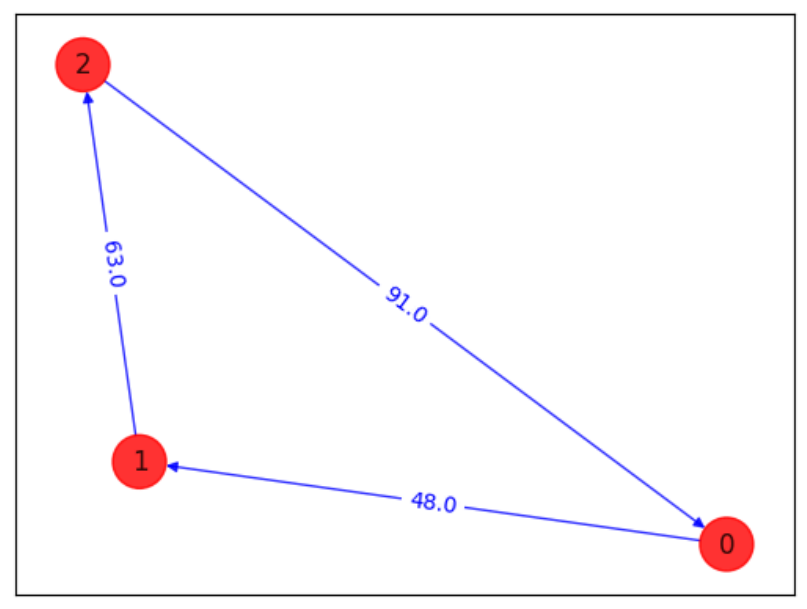
Get the edge labels for the directed graph (G2). The edge labels are stored in a dictionary, where the key is the edge tuple and the value is the weight of the edge.

Draw the edge labels on the graph. The *pos* parameter specifies the position of each node, the *font_color* parameter specifies the color of the edge labels, and the edge_labels parameter specifies the edge labels.

Call the *draw_tsp_solution()* function to visualize the optimal solution for the TSP problem. The function takes the TSP graph (*tsp.graph*), the optimal solution (*best_order*), the list of node colors *(colors)*, and the list of node positions (*pos*) as arguments.

```
[12]: def draw_tsp_solution(G, order, colors, pos):
          G2 = nx.DiGraph()
          G2.add_nodes_from(G)
          n = len(order)
          for i in range(n):
              j = (i + 1) % n
              G2.add_edge(order[i], order[j], weight=G[order[i]][order[j]]["weight"])
          default_axes = plt.axes(frameon=True)
          nx.draw_networkx(
              G2, node_color=colors, edge_color="b", node_size=600, alpha=0.8, ax=default_axes, pos=pos
          )
          edge_labels = nx.get_edge_attributes(G2, "weight")
          nx.draw_networkx_edge_labels(G2, pos, font_color="b", edge_labels=edge_labels)


      draw_tsp_solution(tsp.graph, best_order, colors, pos)
```




In summary, this code snippet creates a directed graph representing the optimal TSP solution, draws the graph with node colors and edge labels, and visualizes the resulting graph.

**Complexity analysis of Brute Force Approach**

- **Time Complexity -** The time complexity is O(n!) where n is the number of nodes present in our graph. This is because we are generating all the possible permutations of the nodes present in our graph.
- **Space Complexity -** The space complexity is O(n) as the function call stack can go to a max depth of n, due to recursive function calls as there are n nodes in our graph. Here n is the number of nodes present in our graph.

## Solving TSP on Classical Computers - Dynamic Programming Approach

Solving the Traveling Salesman Problem (TSP) using the Dynamic Programming approach is a more efficient way compared to the brute-force approach, especially for relatively small instances of the problem. This approach uses a technique called memoization to avoid redundant calculations. Here are the steps involved in the Dynamic Programming approach to solve TSP:

- Consider a function called cost(i) which represents the minimum cost of a path that starts at city 1, ends at city i, and includes all the cities exactly once in between.

- To calculate the total cost of the cycle, add cost(i) to the distance between city i and city 1, denoted as dist(i, 1). This represents the full cost of a tour that starts and ends at city 1 while visiting all other cities exactly once.
- We need to return the minimum of all cost(i) + dist(i, 1) values.

- To calculate the value of cost(i) using Dynamic Programming, we need to establish a recursive relationship based on sub-problems.

- Let us define a term C(S, i) be the cost of the minimum cost path visiting each vertex in set S exactly once, starting at 1 and ending at i. Then,

  If size of S is 2, then S must be {1, i},
  C(S, i) = dist(1, i)
  Else if size of S is greater than 2.
  C(S, i) = min { C(S-{i}, j) + dis(j, i)}, where j belongs to S, j != i and j != 1.

### Solving TSP on Classical Computers - Dynamic Programming Approach - Code Implementation

Following is an implementation of solving TSP by Dynamic Programming approach in C++.

```
#include <bits/stdc++.h>

using namespace std;
```

```
const int n = 4;

int dist[n + 1][n + 1] = {

  { 0, 0, 0, 0, 0 },

  { 0, 0, 10, 15, 20 },

  { 0, 10, 0, 25, 25 },

  { 0, 15, 25, 0, 30 },

  { 0, 20, 25, 30, 0 },

};

int dp[n + 1][1 << (n + 1)];

int solve(int i, int nodes) {

  if (nodes == ((1 << i) | 3)){

     return dist[1][i];

  }

  if (dp[i][nodes] != 0){

     return dp[i][nodes];

  }

  int res = INT_MAX/100;

  for (int j = 1; j <= n; j++){

     if ((nodes & (1 << j)) && j != i && j != 1){

        res = min(res, solve(j, nodes & (~(1 << i)))+ dist[j][i]);

     }
```

```
  }

  return dp[i][nodes] = res;

}

int main() {

  int ans = INT_MAX;

  for (int i = 1; i <= n; i++){

     ans = min(ans, solve(i, (1 << (n + 1)) - 1)+ dist[i][1]);

  }

  cout<<"min cost is  = "<<ans;

}
```

**Complexity analysis of Dynamic Programming Approach**

- **Time Complexity -** The time complexity is $O(n^2*2^n)$ where n is the number of nodes present in our graph. There are $O(n*2^n)$ subproblems and for each subproblem we are running a for loop of O(n) length, hence the total time complexity comes out to be $O(n^2*2^n)$
- **Space Complexity -** The space complexity is $O(n*2^n)$ where n is the number of nodes present in our graph. This is because the function call stack can go to a depth of $O(n*2^n)$ as there are at most $O(n*2^n)$ subproblems, and each one takes linear time to solve.

## Solving TSP on Quantum Computers

Solving the Traveling Salesman Problem (TSP) on a quantum computer involves converting TSP to a combinatorial optimization problem and then mapping it to an Ising problem and then using the Quantum Approximate Optimization Algorithm (QAOA) to find an approximate solution. The steps involved are as follows-

1. Formulate the problem as a combinatorial optimization problem.

2. Map the problem to an Ising problem and calculate the Ising Hamiltonian.

3. Use Quantum Approximation Optimization Algorithm(QAOA) to find the lowest energy state of the Ising Hamiltonian.
4. QAOA uses a quantum circuit to approximate the ground state (lowest energy state) of the Ising Hamiltonian, which corresponds to the optimal solution of the optimization problem.

**Formulation of TSP as  a combinatorial optimization problem.**

Our objective is to discover the most efficient Hamiltonian cycle within a given graph G, consisting of nodes denoted as V and edges represented as E. So, G = (V,E)

Let n refers to the total number of nodes, which can be expressed as n = |V| and the distances between nodes is specified as $w_{ij}$(the distance from node i to node j).

To describe a Hamiltonian cycle, we introduce a variable $x_{ip}$, where 'i' signifies a node, and 'p' signifies its position within a potential cycle. This variable can take on a value of 1 when the solution designates node 'i' at position 'p' in the cycle.

We require that every node can only appear once in the cycle, and for each time a node has to occur. This leads us to the two constraints -

$$\sum_i x_{i,p} = 1 \ \ \forall p$$

$$\sum_p x_{i,p} = 1 \ \ \forall i.$$

For nodes in our prospective ordering, if $x_{i,p}$ and $x_{j,p+1}$ are both 1, then there should be an energy penalty if nodes i and j are not connected in the graph. The form of this penalty is -

$$\sum_{i,j \notin E} \sum_p x_{i,p} x_{j,p+1} > 0,$$

where it is assumed the boundary condition of the Hamiltonian cycles (p=N) ≡ (p=0). The distance that needs to be minimized is

$$C(\mathbf{x}) = \sum_{i,j} w_{ij} \sum_{p} x_{i,p} x_{j,p+1}.$$

Putting this all together in a single objective function to be minimized, we get the following:

$$C(\mathbf{x}) = \sum_{i,j} w_{ij} \sum_{p} x_{i,p} x_{j,p+1} + A \sum_{p} \left(1 - \sum_{i} x_{i,p}\right)^2 + A \sum_{i} \left(1 - \sum_{p} x_{i,p}\right)^2,$$

Having formulated the TSP problem as a combinatorial optimization problem, the next step is to map this problem to an Ising problem and use QAOA to derive the lowest energy state of the Ising Hamiltonian to get the optimal solution of the TSP problem. This part will be covered by us in C2.

# VI. Code Implementation of TSP formulated as a combinatorial optimization problem

```
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from ibm_quantum_widgets import *

# qiskit-ibmq-provider has been deprecated.
# Please see the Migration Guides in https://ibm.biz/provider_migration_guide for more detail.
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler, Estimator, Session, Options

# Loading your IBM Quantum account(s)
service = QiskitRuntimeService(channel="ibm_quantum")

# Invoke a primitive. For more details see https://qiskit.org/documentation/partners/qiskit_ibm_runtime/tutorials.html
# result = Sampler("ibmq_qasm_simulator").run(circuits).result()
```

The above snippet adds necessary packages

The “QuantumCircuit” and “transpile” packages are used to create and optimize circuits while “qiskit.tools.jupyter” and “qiskit.visualization" help in working with jupyter notebook.

“Qiskit_ibm_runtime” help in connecting to IBM Quantum’s runtime services

```
def draw_graph(G, colors, pos):
    default_axes = plt.axes(frameon=True)
    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")
    nx.draw_networkx_edge_labels(G, pos=pos, edge_labels=edge_labels)
```

The above function is designed to draw a graph using the NetworkX library and visualize it with specified node colors and positions

**Parameters:**

**G**: This parameter represents a NetworkX graph object. NetworkX is a Python library for working with graphs.

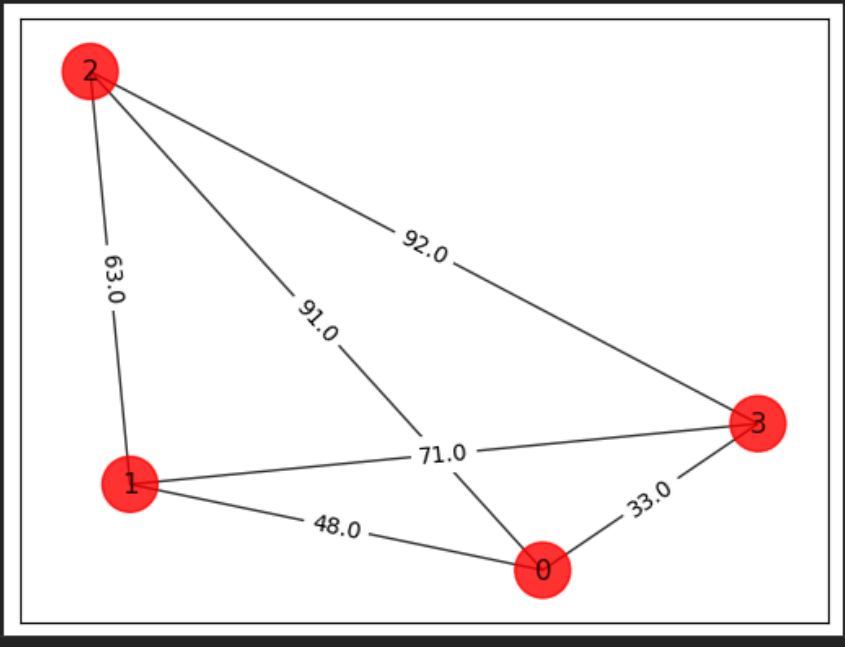
**colors**: This parameter is a list or array that specifies the color of each node in the graph G. The length of this list should be equal to the number of nodes in the graph.

**pos**: This parameter is a dictionary that specifies the positions of nodes in the graph. It maps node identifiers to their respective positions in a two-dimensional space.

```
[4]: # Generating a graph of 4 nodes
n = 4
num_qubits = n**2
tsp = Tsp.create_random_instance(n, seed=123)
adj_matrix = nx.to_numpy_array(tsp.graph)
print("distance\n", adj_matrix)

colors = ["r" for node in tsp.graph.nodes]
pos = [tsp.graph.nodes[node]["pos"] for node in tsp.graph.nodes]
draw_graph(tsp.graph, colors, pos)
```

```
distance
 [[ 0. 48. 91. 33.]
 [48.  0. 63. 71.]
 [91. 63.  0. 92.]
 [33. 71. 92.  0.]]
```




The above function is generating a random instance for TSP and visualizing it as a graph and also displays it in the form of a adjacency matrix
n = is the number of nodes
num_qubits = is the number of qubits. It seems to use a square of the number of nodes as a heuristic for the number of qubits.
Adj_matrix =  converts the TSP graph into an adjacency matrix

```
qp = tsp.to_quadratic_program()
print(qp.prettyprint())
```

```
Problem name: TSP

Minimize
  48*x_0_0*x_1_1 + 48*x_0_0*x_1_3 + 91*x_0_0*x_2_1 + 91*x_0_0*x_2_3
  + 33*x_0_0*x_3_1 + 33*x_0_0*x_3_3 + 48*x_0_1*x_1_0 + 48*x_0_1*x_1_2
  + 91*x_0_1*x_2_0 + 91*x_0_1*x_2_2 + 33*x_0_1*x_3_0 + 33*x_0_1*x_3_2
  + 48*x_0_2*x_1_1 + 48*x_0_2*x_1_3 + 91*x_0_2*x_2_1 + 91*x_0_2*x_2_3
  + 33*x_0_2*x_3_1 + 33*x_0_2*x_3_3 + 48*x_0_3*x_1_0 + 48*x_0_3*x_1_2
  + 91*x_0_3*x_2_0 + 91*x_0_3*x_2_2 + 33*x_0_3*x_3_0 + 33*x_0_3*x_3_2
  + 63*x_1_0*x_2_1 + 63*x_1_0*x_2_3 + 71*x_1_0*x_3_1 + 71*x_1_0*x_3_3
  + 63*x_1_1*x_2_0 + 63*x_1_1*x_2_2 + 71*x_1_1*x_3_0 + 71*x_1_1*x_3_2
  + 63*x_1_2*x_2_1 + 63*x_1_2*x_2_3 + 71*x_1_2*x_3_1 + 71*x_1_2*x_3_3
  + 63*x_1_3*x_2_0 + 63*x_1_3*x_2_2 + 71*x_1_3*x_3_0 + 71*x_1_3*x_3_2
  + 92*x_2_0*x_3_1 + 92*x_2_0*x_3_3 + 92*x_2_1*x_3_0 + 92*x_2_1*x_3_2
  + 92*x_2_2*x_3_1 + 92*x_2_2*x_3_3 + 92*x_2_3*x_3_0 + 92*x_2_3*x_3_2

Subject to
  Linear constraints (8)
    x_0_0 + x_0_1 + x_0_2 + x_0_3 == 1  'c0'
    x_1_0 + x_1_1 + x_1_2 + x_1_3 == 1  'c1'
    x_2_0 + x_2_1 + x_2_2 + x_2_3 == 1  'c2'
    x_3_0 + x_3_1 + x_3_2 + x_3_3 == 1  'c3'
    x_0_0 + x_1_0 + x_2_0 + x_3_0 == 1  'c4'
    x_0_1 + x_1_1 + x_2_1 + x_3_1 == 1  'c5'
    x_0_2 + x_1_2 + x_2_2 + x_3_2 == 1  'c6'
    x_0_3 + x_1_3 + x_2_3 + x_3_3 == 1  'c7'

  Binary variables (16)
    x_0_0 x_0_1 x_0_2 x_0_3 x_1_0 x_1_1 x_1_2 x_1_3 x_2_0 x_2_1 x_2_2 x_2_3
    x_3_0 x_3_1 x_3_2 x_3_3
```

Above snippet uses the Qiskit optimization library to convert a Traveling Salesman Problem (TSP) instance into a Quadratic Program (QP) and then printing a human-readable representation of the QP.

Minimize: specifies the objective function that needs to be minimized in the optimization problem

The expression 48*x_0_0*x_1_1 + 48*x_0_0*x_1_3 + 91*x_0_0*x_2_1 + 91*x_0_0*x_2_3 represents a part of the objective function in the quadratic program (QP) for the Traveling Salesman Problem (TSP). x_0_0, x_1_1, x_1_3, x_2_1, and x_2_3 are binary decision variables. In the context of the TSP, these variables typically represent whether a specific path or edge

between two cities (nodes) in the TSP graph is chosen (1) or not chosen (0)

x_0_0 represents the decision to include the edge from city 0 to itself in the tour.

x_1_1 represents the decision to include the edge from city 1 to itself in the tour.

x_1_3 represents the decision to include the edge from city 1 to city 3 in the tour.

x_2_1 represents the decision to include the edge from city 2 to city 1 in the tour.

x_2_3 represents the decision to include the edge from city 2 to city 3 in the tour.

The entire expression 48*x_0_0*x_1_1 + 48*x_0_0*x_1_3 + 91*x_0_0*x_2_1 + 91*x_0_0*x_2_3 represents a part of the objective function that calculates the total distance of the tour based on the selected edges

Subject to: The "Subject to" section specifies the constraints that the optimization problem must satisfy. In this case, it includes linear constraints and binary variables.

The linear constraints (e.g., x_0_0 + x_0_1 + x_0_2 + x_0_3 == 1 'c0') ensure that each city is visited exactly once. Each constraint specifies that the sum of binary variables for a particular city (e.g., x_0_0, x_0_1, x_0_2, and x_0_3) must equal 1. The 'c0' is a label for the constraint.

## Solving for the optimized value of the objective function

The ground state of the Ising Hamiltonian corresponds to the optimal solution of the objective function. We need to compute the energy associated with the ground state and the state itself. These can be obtained by computing the eigenstates and eigenvectors associated with the Ising Operator.

**Eigenstates/Eigenvectors**

Eigenstates are to operators what Eigenvalues are to matrices. For example, if $|\psi\rangle$ is an eigenstate of the operator A, then $A|\psi\rangle=\lambda|\psi\rangle$, where $\lambda$ is the corresponding eigenvalue. The eigenstates represent the quantum states of a system. The ground state of the system corresponds to the eigenvector associated with the lowest eigenvalue.

**Eigenvalues**

The eigenvalues of a Hamiltonian represent the possible energy levels of the quantum system.

Each eigenvalue corresponds to a specific energy state that the system can occupy. The ground state of a system has the lowest eigenvalue and excited states correspond to have higher eigenvalues. The differences between eigenvalues are associated with the energy transitions that occur in the system.

**Calculating the Eigenstates and Eigenvalues**

There are two ways to calculate the eigenstate and the eigenvectors of the Ising Operator.

- NumpyEigenSolver - Used for finding the minimum eigenvalue of a given matrix using NumPy, which is a popular numerical computing library for Python. Takes the Ising Operator as an input and provides the minimum eigenvalue as the output.
- SamplingVQE - Uses an optimizer (SPSA) to find the minimum energy associated with the quantum state. SPSA is a gradient descent method for optimizing systems with multiple unknown parameters.

**Using NumpyEigenSolver**

```
# Computing the minimum eigenvalue and thus the  solution of the Objective function
eS = NumPyMinimumEigensolver()
eigenvalue_result = eS.compute_minimum_eigenvalue(IsingOperator)
x = tsp.sample_most_likely(eigenvalue_result.eigenstate)
print("energy:", eigenvalue_result.eigenvalue.real)
print("feasible:", qubo.is_feasible(x))
solver = MinimumEigenOptimizer(numpyEigenSolver())
result = solver.solve(qubo)
print(result.prettyprint())
```

```
energy: -7379.0
feasible: True
objective function value: 202.0
variable values: x_0_0=1.0, x_0_1=0.0, x_0_2=0.0, x_1_0=0.0, x_1_1=1.0, x_1_2=0.0, x_2_0=0.0, x_2_1=0.0, x_2_2=1.0
status: SUCCESS
```

In this method, we create an instance of NumpyMinimumEigensolver and compute the minimum eigenvalue associated with the Ising form of our problem. Then we find the most likely state at this lowest energy level and check if it is feasible or not. We find that, it indeed is feasible. If we found it was not feasible, we would need to refine the quantum algorithm, adjust parameters, or, in some cases, run the quantum algorithm multiple times to increase the chance of obtaining a feasible solution. Finally, we compute the value of the binary variables and substituting their value in the objective function gives us the objective function value as 202.

**Using SamplingVQE**

```
: optimizer = SPSA(maxiter=300)
  ry = TwoLocal(IsingOperator.num_qubits, "ry", "cz", reps=5, entanglement="linear")
  vqe = SamplingVQE(sampler=Sampler(), ansatz=ry, optimizer=optimizer)
  eigenvalue_result = vqe.compute_minimum_eigenvalue(IsingOperator)
  print("energy:", eigenvalue_result.eigenvalue.real)
  x = tsp.sample_most_likely(eigenvalue_result.eigenstate)
  print("feasible:", qubo.is_feasible(x))
  z = tsp.interpret(x)
  print("solution: [0,1,2]")
  print("solution objective:", tsp.tsp_value(z, adj_matrix))
  draw_tsp_solution(tsp.graph, z, colors, pos)
```

```
energy: -7326.02469952184
feasible: True
solution: [0,1,2]
solution objective: 202.0
```

In this method, SPSA optimizer is used to find the lowest energy state of the Ising system. We also need to create a parametrized quantum circuit to prepare the trial state and pass both the optimizer and the circuit to SampleVQE instance along with the sampler. The sampler primitive is used to sample the circuits. Sampling means calculating quasi-probabilities of bitstrings from quantum circuits.

Next steps are the same as in the previous approach. We calculate the energy of the lowest energy state, find the most likely solution, check its feasibility and print the solution along with the Objective function value.

## VII. D-Wave Solution

The TSP involves finding a minimal-cost roundtrip through a fully connected graph. The quantum approach leverages the principles of adiabatic quantum computation, where qubits evolve from an initial ground state to a target state.

D-Wave's quantum computers implement adiabatic quantum computation through quantum annealing, with the Hamiltonian constrained to an Ising model. The energy associated with the target state is expressed using biases (hi) and coupling constants (Jij) in the Ising Hamiltonian. Quantum annealing involves gradually transitioning from the H0 state, achieved with a transverse magnetic field, to the H1 state, allowing qubits to align and minimize the energy.

$$H(t) = (1-t)H_0 + tH_1$$

$$E_{Ising} = -\sum_{i} h_i s_i - \sum_{i} \sum_{j,j \neq i} J_{ij} s_i s_j$$

The paper notes that a simple change of variables transforms the Ising model into a Quadratic Unconstrained Binary Optimization (QUBO) problem, expressed as minimizing the quadratic form x^TQx. D-Wave's devices are specialized to address QUBO problems efficiently.

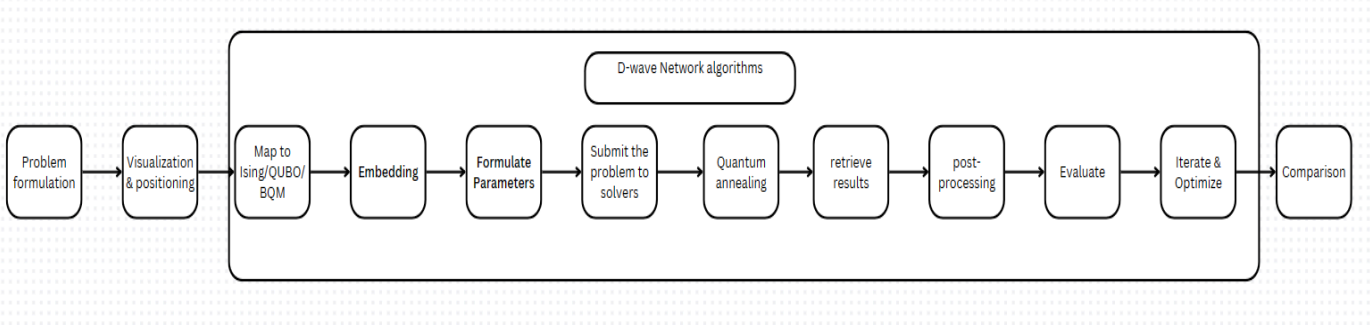
In conclusion, the study presents a quantum solution to the TSP using D-Wave quantum computers, utilizing adiabatic quantum computation and quantum annealing to optimize the Ising Hamiltonian, ultimately providing a new approach to solving complex optimization problems.

Objective function for TSP can be written as

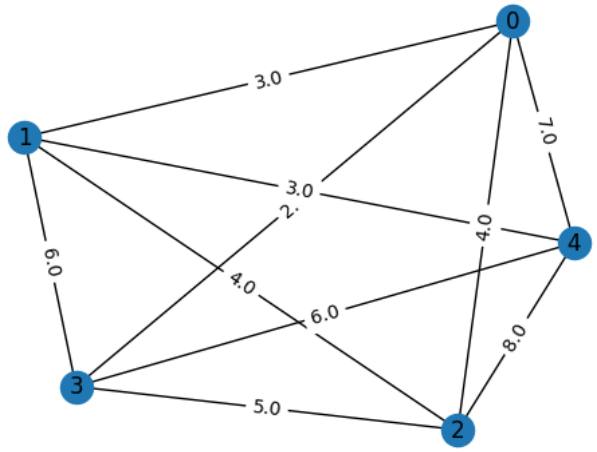
$$f = \sum M. * X = \sum_{i=0}^{i<n} \sum_{j=i+1}^{j<n} \left(M_{ij} + M_{ji}\right) x_{ij} = \sum_{k=0}^{k<m} B_k b_k$$

The Dwave_networkx provides algorithms for TSP solution and internal working includes formulation of QUBO, using the method of Lagrange multipliers to absorb the constraints into the objective function. The approach involves addressing challenges such as minor embedding into the physical qubit graph and parameter tuning. The Lagrange multiplier, chain strength, and annealing time are crucial parameters affecting the quantum solution.

Proposed Methodology




**TSP problem 1:**




1. Simulated annealing on classical computers

```
import time

# get the start time
st = time.time()
#use(classical) simulated annealing
%time route = dnx.traveling_salesperson(G, dimod.SimulatedAnnealingSampler(), start=0)
print("Route found with simulated annealing:", route)

# get the end time
et = time.time()

# get the execution time
elapsed_time = et - st
print('Execution time:', elapsed_time, 'seconds')
```

```
CPU times: user 3.78 s, sys: 13.4 ms, total: 3.8 s
Wall time: 3.95 s
Route found with simulated annealing: [0, 3, 2, 4, 1]
Execution time: 3.94789981842041 seconds
```

D-Wave quantum annealers are primarily designed for quantum computing, and the SimulatedAnnealingSolver is a classical algorithm implemented in the context of D-Wave's Ocean software suite.

The traveling_salesperson function in D-Wave's Ocean software (dnx module) is designed to solve the Traveling Salesperson Problem (TSP) using the D-Wave quantum processor. The TSP is a well-known combinatorial optimization problem where the goal is to find the shortest possible route that visits a set of cities and returns to the starting city.

Defines a Quadratic Unconstrained Binary Optimization (QUBO) problem where the ground states correspond to the minimum routes in a given graph. The goal is to find a cycle in the graph that visits each node exactly once, with the minimum total edge weight. The function utilizes a binary quadratic model sampler, which is a process capable of sampling low energy states in models defined by an Ising equation or QUBO. The sampler is expected to have 'sample_qubo' and 'sample_ising' methods. The user can provide a NetworkX graph (G) representing the problem, a sampler, an optional Lagrange parameter (lagrange) to balance constraints and objectives, an edge attribute containing weights (default is 'weight'), a starting node (start), and additional parameters for the sampler. The function returns a list of nodes representing the order to be visited on the minimum route.

2. Using Exact Solver

```
] %time exact_route = dnx.traveling_salesperson(G, dimod.ExactSolver(), start=0)
  print("Route found with exact solver(brute force)", exact_route)

  CPU times: user 1min 36s, sys: 2.11 s, total: 1min 38s
  Wall time: 2min 3s
  Route found with exact solver(brute force) [0, 3, 4, 1, 2]

] total_dist=0
  for idx, node in enumerate(route[:-1]):
    dist=data[route[idx+1]][route[idx]]
    total_dist +=dist

] print("Total distance (Without return):" , total_dist)

  Total distance (Without return): 15.0

  return_distance= data[route[0]][route[-1]]
  print("Distance between start and end", return_distance)

  Distance between start and end 4.0
```

The ExactSolver in D-Wave's Ocean software (dwave.system module) is a classical solver used for solving discrete optimization problems exactly. Unlike quantum processors, which leverage quantum effects to explore solution spaces, the ExactSolver uses classical algorithms to find the optimal solution for a given optimization problem. Impractical as it calculates all possible combinations of paths.

3. Client Initialization and obtaining all solvers

```
from dwave.cloud.client import Client

token = "[redacted]"

client = Client(token=token)
print(client.get_solvers())
```

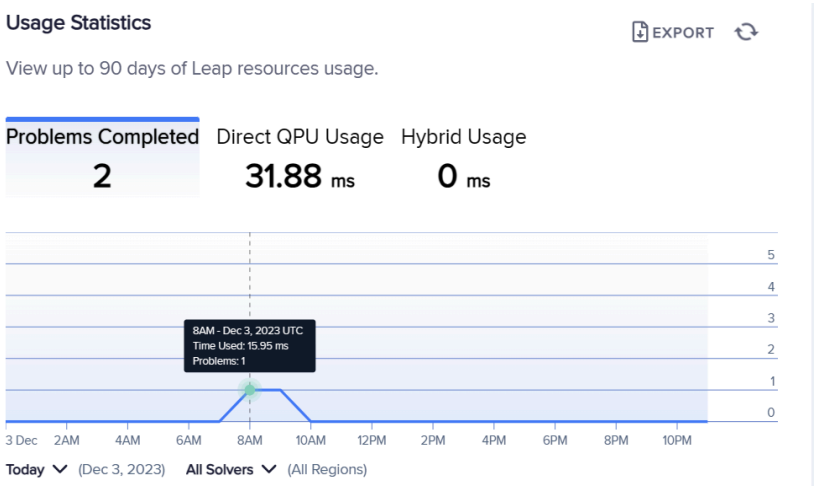
```
[BQMSolver(id='hybrid_binary_quadratic_model_version2'), DQMSo
```

4. Solving on D-wave QPU

```
%time embedded_route = dnx.traveling_salesperson(G, EmbeddingComposite(DWaveSampler(token=token)), start=0)
print("route with Dwave annealer:", embedded_route)
```

```
CPU times: user 4.84 s, sys: 86.6 ms, total: 4.92 s
Wall time: 7.53 s
route with Dwave annealer: [0, 3, 2, 1, 4]
```

DWaveSampler is a class that represents a connection to a D-Wave quantum processing unit (QPU). It allows you to submit problems to the D-Wave QPU for quantum annealing, which is a quantum optimization process aimed at finding the lowest energy state (or states) corresponding to the optimal solution of a given problem.
EmbeddingComposite is a composite sampler that combines a problem's logical qubits (from a higher-level problem representation) with the physical qubits available on the D-Wave QPU. It handles the process of embedding a problem onto the qubits of the D-Wave QPU, taking care of the necessary qubit mappings.



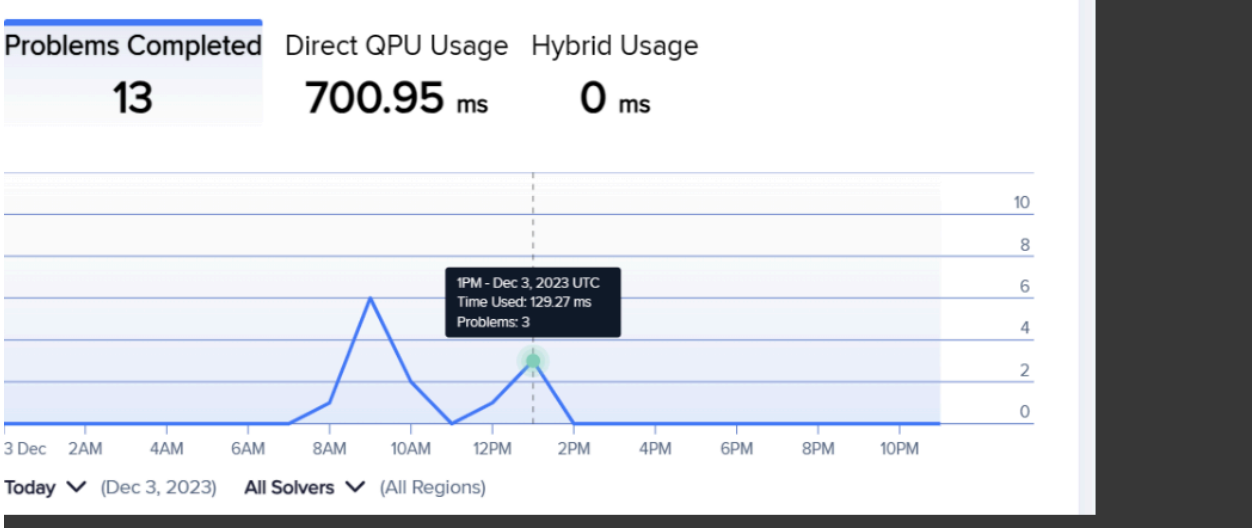
```
[ ]  total_dist=0
     for idx, node in enumerate(embedded_route2[:-1]):
       dist=data[embedded_route[idx+1]][embedded_route2[idx]]
       total_dist +=dist
     print("Total distance exact_route (Without return):" , total_dist)
     return_distance= data[embedded_route2[0]][embedded_route2[-1]]
     print("Distance between exact_route start and end", return_distance)
     distance= total_dist+ return_distance
     print("total including return: exact_route", distance)
```

```
Total distance exact_route (Without return): 17.0
Distance between exact_route start and end 2.0
total including return: exact_route 19.0
```

5. Using num_reads

```
%time embedded_big_route_reads = dnx.traveling_salesperson(G,  EmbeddingComposite(DWaveSampler(token=token)), num_reads=1000, start=0)
print("Route found with simulated annealing:", embedded_big_route_reads)
```

```
CPU times: user 3.61 s, sys: 54.9 ms, total: 3.67 s
Wall time: 5.6 s
Route found with simulated annealing: [0, 3, 4, 1, 2]
```




Trying to control lagrange

```
sampler = EmbeddingComposite(DWaveSampler(token=token))
```

```
response= sampler.sample_qubo(qubo)
sample= response.first.sample
```

```
response
```

```
SampleSet(rec.array([([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1], -170., 1, 0.)],
          dtype=[('sample', 'i1', (25,)), ('energy', '<f8'), ('num_occurrences', '<i8'), ('chain_break_fraction', '<
(2, 4), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]), {'timing': {'qpu_sampling_
'qpu_access_overhead_time': 214651.39, 'qpu_programming_time': 15781.57, 'qpu_delay_time_per_sample': 20.54, 'total_
'BINARY')
```

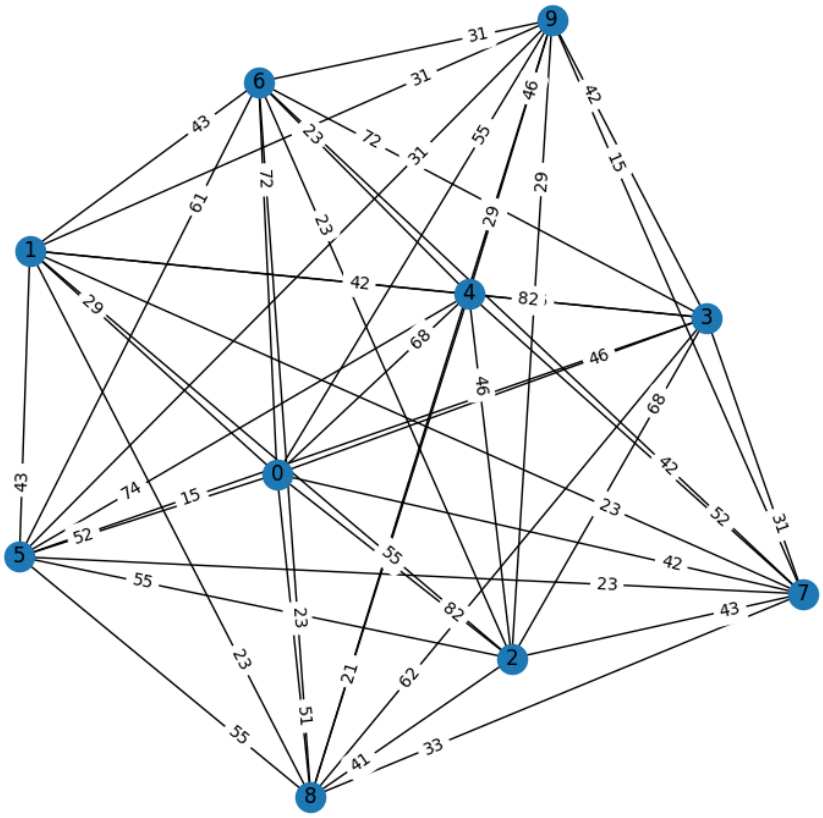
```
response.info
```

```
{'timing': {'qpu_sampling_time': 97.04,
  'qpu_anneal_time_per_sample': 20.0,
  'qpu_readout_time_per_sample': 56.5,
  'qpu_access_time': 15878.61,
  'qpu_access_overhead_time': 214651.39,
  'qpu_programming_time': 15781.57,
  'qpu_delay_time_per_sample': 20.54,
  'total_post_processing_time': 98.0,
  'post_processing_overhead_time': 98.0},
 'problem_id': '0cb08278-2ccf-4a61-9d7f-382c96d0deb4'}
```

**TSP Problem 2:**

NetworkX graph (G) is constructed from a pandas DataFrame representing an adjacency matrix (data10). The positions of nodes in the graph are then determined using the Spring layout algorithm (nx.spring_layout), with a specified random seed (seed). The nodes, edges, and weights of the graph are extracted for further visualization.

The code utilizes Matplotlib to create a figure with a size of 10 by 10 inches (plt.figure(figsize=(10,10))). The axis is turned off (plt.axis('off')) to remove the axis labels and ticks. Subsequently, the graph is visualized using NetworkX's draw_networkx function, with the node positions determined earlier (pos). Nodes are labeled (with_labels=True), and the edges are drawn with weights displayed as labels at a specific position (label_pos=0.25) using draw_networkx_edge_label.




We applied the same process of problem 1 to problem 2 as well.

# VIII. Comparison

| $n$ | Number of paths | Time ($1\mu s/chemin$) |
|---|---|---|
| 5 | 12 | $12\mu s$ |
| 10 | 181, 440 | $0.18s$ |
| 15 | $4.359\times10^{10}$ | $12h$ |
| 20 | $6.082\times10^{16}$ | 1, 928 years |
| 61 | $4.160\times10^{81}$ | $13.19\times10^{67}$ years |

Fig:Number of paths and estimated calculation time as a function of the number of vertices. The number

of paths is comparable with the estimated number of atoms in the visible universe, which is 1080, and the time required to enumerate all solutions is similar to the age of the universe, which is 13.798 ± 0.037 × 109 years

| Number of cities | Brute Force | Gate quantum computer |
|---|---|---|
| 3 | 0.4ms | 14s |
| 6 | 0.001 secs | Crashed |

Table: Comparison Of brute force performance vs Gate model

| Number of cities | Simulated Annealing (using Classical computer) | Exact Solver | D-Wave Sampler(Direct QPU Usage(CPU time in notebook)) | D-Wave Sampler(num_reads=1000) (Direct QPU Usage) |
|---|---|---|---|---|
| 5 | 4.97s | 1m 38s | 31.88ms (3.67s-5.45s) | 129.27ms |
| 10 | 36.2s | None | 31.89s (1min 36s) | 226.25ms |

Table: Comparison of classical solver vs quantum solution

# IX. Future Plan

We will use the QAOA algorithm to find the lowest energy state of the Ising Hamiltonian which will correspond to the solution of the TSP formulated as a quadratic optimization problem.

D-Wave:
We may Hybrid Solver and all other Advantage_systems for the execution. Instead of QUBO, BQM a mix of Ising and QUBO can be used to evaluate tsp solution.

# X. Conclusion

Quantum computing utilizes quantum mechanics to tackle complex problems beyond classical computers' capabilities. Key advantages include parallel processing through qubits' superposition and entanglement, enabling linked qubits to share the same fate. Despite being in early stages, quantum computing achieved milestones, such as Google's quantum supremacy in 2019.

Promisingly, quantum computing excels in solving optimization problems, exemplified by the Traveling Salesman Problem (TSP). Quantum algorithms like QAOA show potential in efficiently exploring multiple routes simultaneously, outperforming classical counterparts.

In conclusion, the Traveling Salesman Problem (TSP) has been successfully addressed using Quantum Variational Eigensolver (VQE) on gate model quantum computers, where the problem is transformed from QUBO to Ising. Additionally, D-Wave annealers have provided alternative solutions. Noteworthy observations include the contrast in handling constraints: classical solvers strictly adhere, while quantum solutions turn constraints into soft constraints that may be violated.

At first glance, neither gate model quantum computers nor D-Wave Quantum Processing Units (QPUs) exhibit a clear quantum advantage over classical solvers. However, a notable advantage emerges as the number of cities increases, with the time taken showing at most linear growth rather than exponential.

As technology advances, there is potential for quantum solutions to outperform classical methods, especially as quantum computing matures. The prospect of overcoming classical limitations in tackling complex optimization problems, like the TSP, remains an exciting avenue for future exploration and development.

While still developing, quantum computing's transformative potential spans various industries. Recent accessibility enhancements, exemplified by tools like Qiskit, enable broader exploration and experimentation. Quantum computing is no longer a distant concept but an emerging reality with the capacity to revolutionize diverse fields.

Supplemenatry Resources: TSP_DWAVE.ipynb - Colaboratory (google.com)
PPT:Study of TSP on Quantum Computers(C3) - Google Slides

# XI. References


- Quantum logic gate - Wikipedia
- Parallel quantum annealing | Scientific Reports (nature.com)
- 2212.10990.pdf (arxiv.org) | Frontiers | NP-hard but no longer hard to solve? Using quantum computing to tackle optimization problems (frontiersin.org)

- [2301.06978] Solving various NP-Hard problems using exponentially fewer qubits on a Quantum Computer (arxiv.org)
- Decomposition Algorithms for Solving NP-hard Problems on a Quantum Annealer | SpringerLink
- Quantum algorithm provides new approach to NP-hard problem – Physics World
- Solving NP-hard Problems with Quantum Annealing | IEEE Conference Publication | IEEE Xplore
- Qiskit - YouTube
- Why the world's toughest maths problems are much harder than a chess puzzle, and well worth US$1m (theconversation.com)
- Au-Yeung R, Chancellor N and Halffmann P (2023), NP-hard but no longer hard to solve? Using quantum computing to tackle optimization problems. Front. Quantum. Sci. Technol. 2:1128576. doi: 10.3389/frqst.2023.1128576
- “Solver Docs” D-Wave. Solver Docs, https://docs.dwavesys.com/docs/latest/guides_solvers.html (accessed 2023).
- Jain S (2021) Solving the Traveling Salesman Problem on the D-Wave Quantum Computer. Front. Phys. 9:760783. doi: 10.3389/fphy.2021.760783
- Feld S, Roch C, Gabor T, Seidel C, Neukart F, Galter I, Mauerer W andLinnhoff-Popien C (2019) A Hybrid Solution Method for the Capacitated Vehicle Routing Problem Using a Quantum Annealer. Front. ICT 6:13. doi: 10.3389/fict.2019.00013
- Advantage Performance white paper