Instance Space Analysis on Quantum Algorithms

Vivek Katial

02/03/2023

Introduction

Background

- Developing a framework to robustly evaluate quantum algorithms using Instance Space Analysis
- Explore the tools needed to develop Quantum Algorithms:
  - Adiabatic Quantum Algorithms (year 1)
  - Universal Gate Based Quantum Algorithms (year 2)
  - Explore various optimization problems, initialization techniques and instance classes (year 3)

Agenda

- Optimization Problems

Agenda

- Optimization Problems
- Quantum Algorithms

Agenda

- Optimization Problems
- Quantum Algorithms
- Methodology

Agenda

- Optimization Problems
- Quantum Algorithms
- Methodology
- Results

Agenda

- Optimization Problems
- Quantum Algorithms
- Methodology
- Results
- Next Steps

Agenda

- Optimization Problems
- Quantum Algorithms
- Methodology
- Results
- Next Steps
- Research Plan

Optimisation Problems

- MAXCUT (will explore today)
- Other studied problems:
  - Traveling Salesperson Problem (TSP) and Vehicle Routing (VRP)
  - 3SAT - Exact Cover

Quantum Algorithms

- **Adiabatic Quantum Computing**

Quantum Algorithms

- **Adiabatic Quantum Computing**
  - AQC is a specific type of quantum computing that uses adiabatic evolution to solve optimization problems [1].

Quantum Algorithms

- **Adiabatic Quantum Computing**
  - AQC is a specific type of quantum computing that uses adiabatic evolution to solve optimization problems [1].
  - AQC relies on the natural evolution of the quantum system, avoiding the need for precise control over individual quantum gates [2].

Quantum Algorithms

- **Adiabatic Quantum Computing**
  - AQC is a specific type of quantum computing that uses adiabatic evolution to solve optimization problems [1].
  - AQC relies on the natural evolution of the quantum system, avoiding the need for precise control over individual quantum gates [2].
- **Gate Based Quantum Computing**

Quantum Algorithms

- **Adiabatic Quantum Computing**
  - AQC is a specific type of quantum computing that uses adiabatic evolution to solve optimization problems [1].
  - AQC relies on the natural evolution of the quantum system, avoiding the need for precise control over individual quantum gates [2].
- **Gate Based Quantum Computing**
  - Uses a series of quantum gates to manipulate the quantum system and perform specific computations [3].

Quantum Algorithms

- **Adiabatic Quantum Computing**
  - AQC is a specific type of quantum computing that uses adiabatic evolution to solve optimization problems [1].
  - AQC relies on the natural evolution of the quantum system, avoiding the need for precise control over individual quantum gates [2].
- **Gate Based Quantum Computing**
  - Uses a series of quantum gates to manipulate the quantum system and perform specific computations [3].
  - Requires precise control over individual quantum gates, which can be challenging to implement [4].

Quantum Algorithms

- **Adiabatic Quantum Computing**
  - AQC is a specific type of quantum computing that uses adiabatic evolution to solve optimization problems [1].
  - AQC relies on the natural evolution of the quantum system, avoiding the need for precise control over individual quantum gates [2].
- **Gate Based Quantum Computing**
  - Uses a series of quantum gates to manipulate the quantum system and perform specific computations [3].
  - Requires precise control over individual quantum gates, which can be challenging to implement [4].
  - Potential for faster performance, as it can use advanced techniques such as error correction and quantum parallelism [3].

MAXCUT

MAXCUT

- Given a graph $G = (V, E)$ with vertices V and edges E.

MAXCUT

- Given a graph $G = (V, E)$ with vertices V and edges E.
- Each edge $(i, j) \in E$ has a weight $w(i, j)$.

MAXCUT

- Given a graph $G = (V, E)$ with vertices V and edges E.
- Each edge $(i, j) \in E$ has a weight $w(i, j)$.
- Partition the vertices into two disjoint sets S and T, such that $S \cup T = V$ and $S \cap T = \emptyset$.

MAXCUT

- Given a graph $G = (V, E)$ with vertices V and edges E.
- Each edge $(i, j) \in E$ has a weight $w(i, j)$.
- Partition the vertices into two disjoint sets S and T, such that $S \cup T = V$ and $S \cap T = \emptyset$.
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at time step t

MAXCUT

- Given a graph $G = (V, E)$ with vertices V and edges E.
- Each edge $(i, j) \in E$ has a weight $w(i, j)$.
- Partition the vertices into two disjoint sets S and T, such that $S \cup T = V$ and $S \cap T = \emptyset$.
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at time step t
  - Our goal is to find a partition that maximizes the MAXCUT value:

MAXCUT

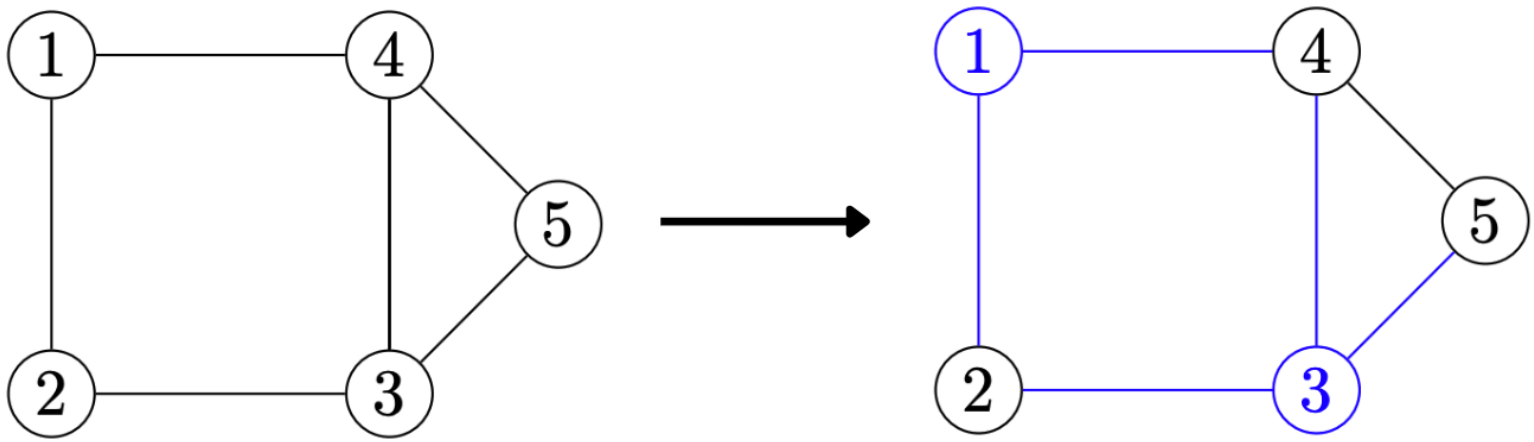
- Given a graph $G = (V, E)$ with vertices V and edges E.
- Each edge $(i, j) \in E$ has a weight $w(i, j)$.
- Partition the vertices into two disjoint sets S and T, such that $S \cup T = V$ and $S \cap T = \emptyset$.
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at time step t
  - Our goal is to find a partition that maximizes the MAXCUT value:
  -

$$C(\mathbf{x}) = \sum_{i,j} w_{i,j} \; x_i(1 - x_j)$$

MAXCUT

- Given a graph $G = (V, E)$ with vertices V and edges E.
- Each edge $(i, j) \in E$ has a weight $w(i, j)$.
- Partition the vertices into two disjoint sets S and T, such that $S \cup T = V$ and $S \cap T = \emptyset$.
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at time step t
  - Our goal is to find a partition that maximizes the MAXCUT value:
  - $$C(\mathbf{x}) = \sum_{i,j} w_{i,j} \; x_i(1 - x_j)$$
- This problem is NP-hard [5]

MAXCUT Formulation



Solution: Two partitions are $S = \{1, 2\}$ and $T = \{2, 4, 5\}$. The size of the cut is 5.

Classical Limitations of MAXCUT

- NP-hard to achieve better approximation ratio than [6]:

Classical Limitations of MAXCUT

- NP-hard to achieve better approximation ratio than [6]:

Classical Limitations of MAXCUT

- NP-hard to achieve better approximation ratio than [6]:

$$\alpha \sim 0.941$$

- Best known classical approximation achieved using *Goemans-Williamson* algorithm [7]

Classical Limitations of MAXCUT

- NP-hard to achieve better approximation ratio than [6]:

$$\alpha \sim 0.941$$

- Best known classical approximation achieved using *Goemans-Williamson* algorithm [7]

Classical Limitations of MAXCUT

- NP-hard to achieve better approximation ratio than [6]:

$$\alpha \sim 0.941$$

- Best known classical approximation achieved using *Goemans-Williamson* algorithm [7]

$$\alpha \sim 0.878$$

Mapping MAXCUT to QUBO

- To solve this problem on a Quantum Computer we need to map it to a QUBO.

Mapping MAXCUT to QUBO

- To solve this problem on a Quantum Computer we need to map it to a QUBO.
- $c_i = \sum_{j=1}^{n} W_{ij}, \;\; Q_{ij} = -W_{ij}$

Mapping MAXCUT to QUBO

- To solve this problem on a Quantum Computer we need to map it to a QUBO.
- $c_i = \sum_{j=1}^{n} W_{ij}, \;\; Q_{ij} = -W_{ij}$

Mapping MAXCUT to QUBO

- To solve this problem on a Quantum Computer we need to map it to a QUBO.
- $c_i = \sum_{j=1}^{n} W_{ij}, \;\; Q_{ij} = -W_{ij}$

$$C(x) = \sum_{i,j=1}^{n} x_i Q_{ij} x_j + \sum_{i}^{n} c_i x_i = x^T Q x + c^T x$$

- We can assign a qubit to each decision variable, make a the following substitution $x_i = \frac{(I - Z_i)}{2}$. This gives us the resulting encoding for a problem Hamiltonian H_c that can be encoded onto a quantum circuit.

Mapping MAXCUT to QUBO

- To solve this problem on a Quantum Computer we need to map it to a QUBO.
- $c_i = \sum_{j=1}^{n} W_{ij}, \;\; Q_{ij} = -W_{ij}$

$$C(x) = \sum_{i,j=1}^{n} x_i Q_{ij} x_j + \sum_{i}^{n} c_i x_i = x^T Q x + c^T x$$

- We can assign a qubit to each decision variable, make a the following substitution $x_i = \frac{(I - Z_i)}{2}$. This gives us the resulting encoding for a problem Hamiltonian H_c that can be encoded onto a quantum circuit.

Mapping MAXCUT to QUBO

- To solve this problem on a Quantum Computer we need to map it to a QUBO.
- $c_i = \sum_{j=1}^{n} W_{ij}, \;\; Q_{ij} = -W_{ij}$

$$C(x) = \sum_{i,j=1}^{n} x_i Q_{ij} x_j + \sum_{i}^{n} c_i x_i = x^T Q x + c^T x$$

- We can assign a qubit to each decision variable, make a the following substitution $x_i = \frac{(I - Z_i)}{2}$. This gives us the resulting encoding for a problem Hamiltonian H_c that can be encoded onto a quantum circuit.

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

QAOA

- The QAOA algorithm is characterised by a unitary $U(\beta, \gamma)$ to prepare a quantum state $|\psi(\beta, \gamma)\rangle$

QAOA

- The QAOA algorithm is characterised by a unitary $U(\beta, \gamma)$ to prepare a quantum state $|\psi(\beta, \gamma)\rangle$
- The goal of the algorithm is to find optimal parameters $(\beta_{opt}, \gamma_{opt})$ such that the quantum state encodes the solution to the problem.

QAOA

- The QAOA algorithm is characterised by a unitary $U(\beta, \gamma)$ to prepare a quantum state $|\psi(\beta, \gamma)\rangle$
- The goal of the algorithm is to find optimal parameters $(\beta_{opt}, \gamma_{opt})$ such that the quantum state encodes the solution to the problem.
- The state is prepared by applying the unitary p times:

QAOA

- The QAOA algorithm is characterised by a unitary $U(\beta, \gamma)$ to prepare a quantum state $|\psi(\beta, \gamma)\rangle$
- The goal of the algorithm is to find optimal parameters $(\beta_{opt}, \gamma_{opt})$ such that the quantum state encodes the solution to the problem.
- The state is prepared by applying the unitary p times:

QAOA

- The QAOA algorithm is characterised by a unitary $U(\beta, \gamma)$ to prepare a quantum state $|\psi(\beta, \gamma)\rangle$
- The goal of the algorithm is to find optimal parameters $(\beta_{opt}, \gamma_{opt})$ such that the quantum state encodes the solution to the problem.
- The state is prepared by applying the unitary p times:

$$|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle = \underbrace{U(\boldsymbol{\beta})U(\boldsymbol{\gamma}) \cdots U(\boldsymbol{\beta})U(\boldsymbol{\gamma})}_{p \text{ times}} |\psi_0\rangle$$

- Where $U(\beta) = e^{-i\beta H_b}$ and $U(\gamma) = e^{-i\gamma H_c}$

QAOA Algorithm

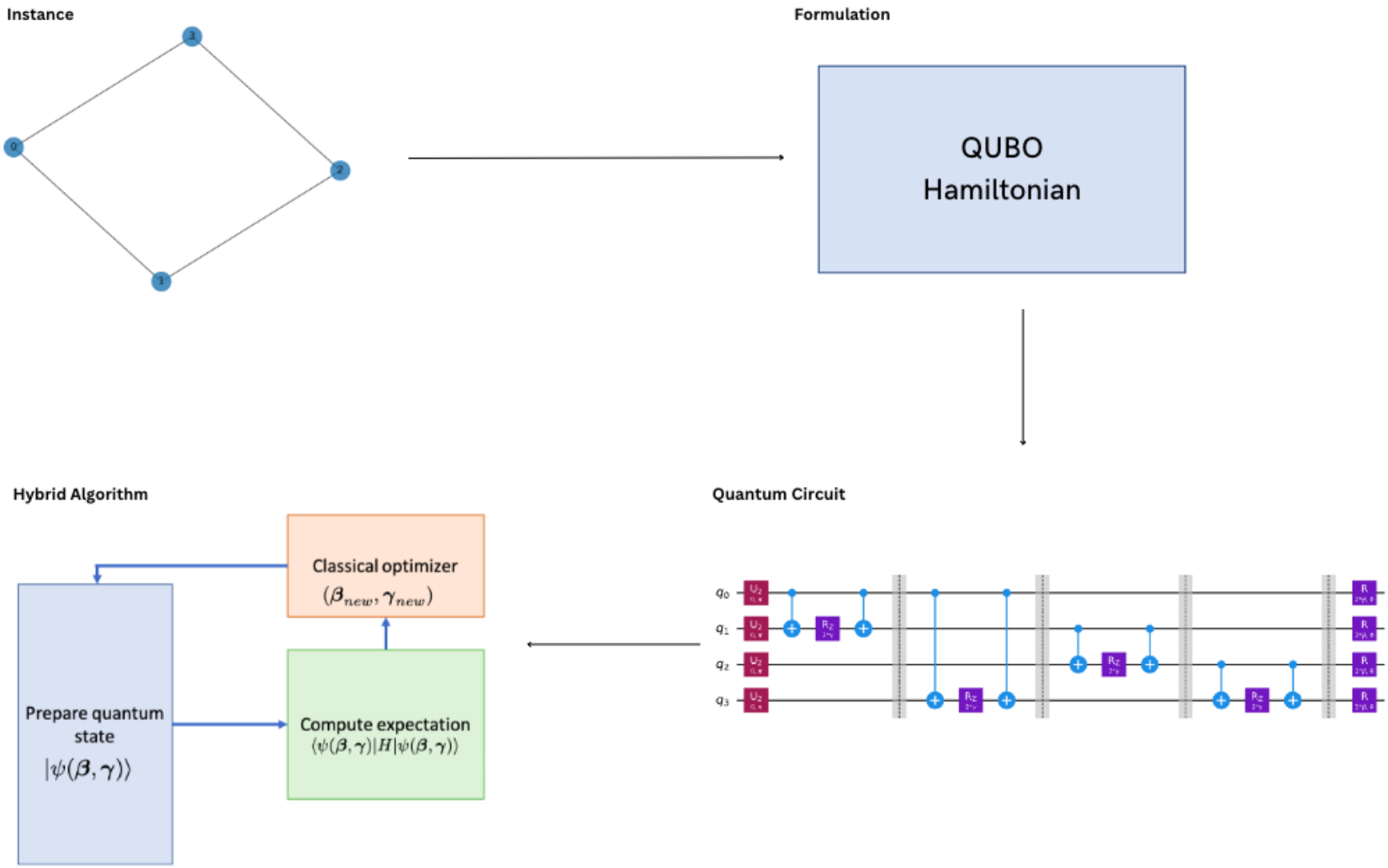
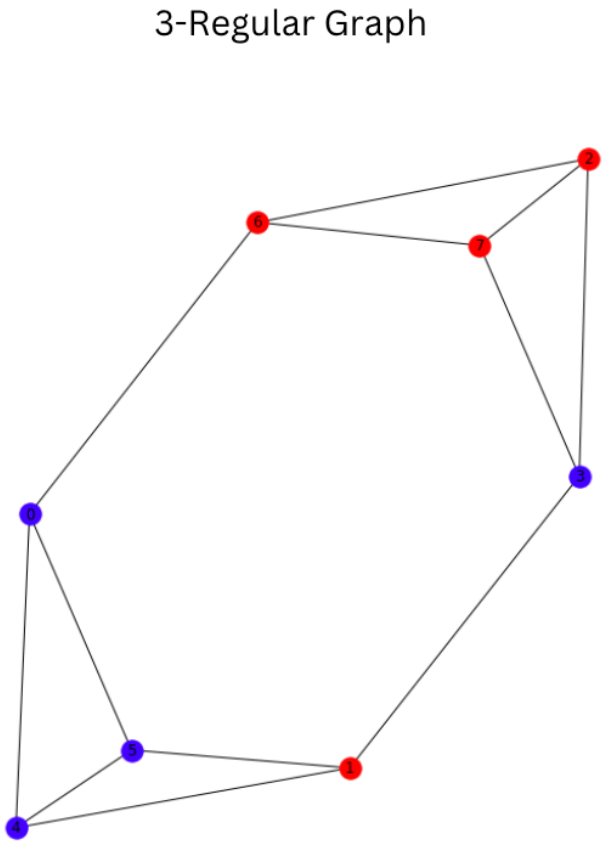
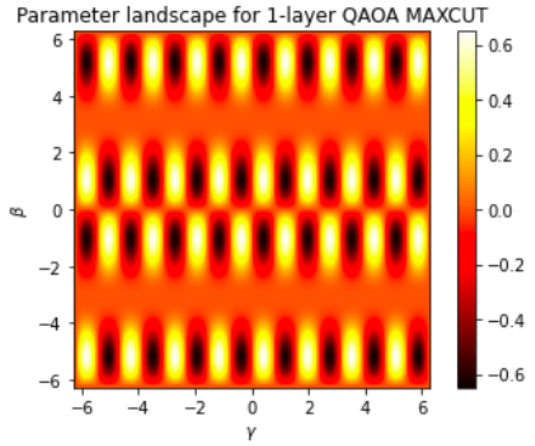


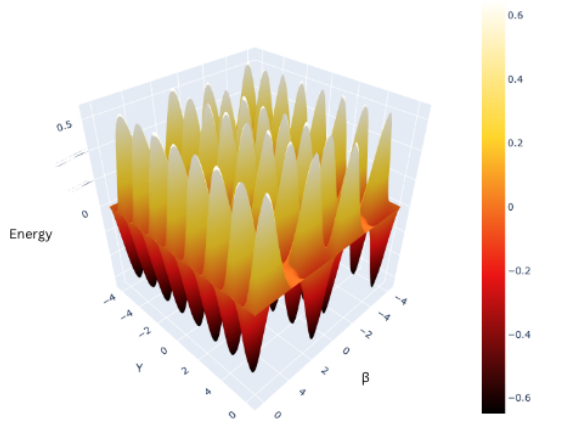
Figure 1: QAOA Strategy

Landscape for $p = 1$

- With QAOA we observe periodicity in γ and β







Applying ISA in the Quantum Context

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)
  - $p = 1, ..., 10$

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)
  - $p = 1, ..., 10$
- QAOA (experimenting with parameter optimization methods)

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)
  - $p = 1, ..., 10$
- QAOA (experimenting with parameter optimization methods)
  - `COBYLA`, `SPSA`, `NELDER_MEAD`

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)
  - $p = 1, ..., 10$
- QAOA (experimenting with parameter optimization methods)
  - `COBYLA`, `SPSA`, `NELDER_MEAD`
- QAOA (experimenting Initialisation Techniques)

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)
  - $p = 1, ..., 10$
- QAOA (experimenting with parameter optimization methods)
  - `COBYLA`, `SPSA`, `NELDER_MEAD`
- QAOA (experimenting Initialisation Techniques)
  - Six methods (outlined further)

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)
  - $p = 1, ..., 10$
- QAOA (experimenting with parameter optimization methods)
  - `COBYLA`, `SPSA`, `NELDER_MEAD`
- QAOA (experimenting Initialisation Techniques)
  - Six methods (outlined further)
- Variational Quantum Algorithms (VQE-2 Local, F-VQE, QAOA Vanilla)

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)
  - $p = 1, ..., 10$
- QAOA (experimenting with parameter optimization methods)
  - `COBYLA`, `SPSA`, `NELDER_MEAD`
- QAOA (experimenting Initialisation Techniques)
  - Six methods (outlined further)
- Variational Quantum Algorithms (VQE-2 Local, F-VQE, QAOA Vanilla)
- **22** unique algorithm variations

Quantum Algorithms and Variations Studied

- QAOA (experimenting with number of layers)
  - $p = 1, ..., 10$
- QAOA (experimenting with parameter optimization methods)
  - `COBYLA`, `SPSA`, `NELDER_MEAD`
- QAOA (experimenting Initialisation Techniques)
  - Six methods (outlined further)
- Variational Quantum Algorithms (VQE-2 Local, F-VQE, QAOA Vanilla)
- **22** unique algorithm variations
- **Can we measure characteristics of instances that make specific algorithms more suited to solve certain graphs?**

Current Literature

- Improvements in QAOA and VQE have largely been limited to exploring three types of graphs:

Current Literature

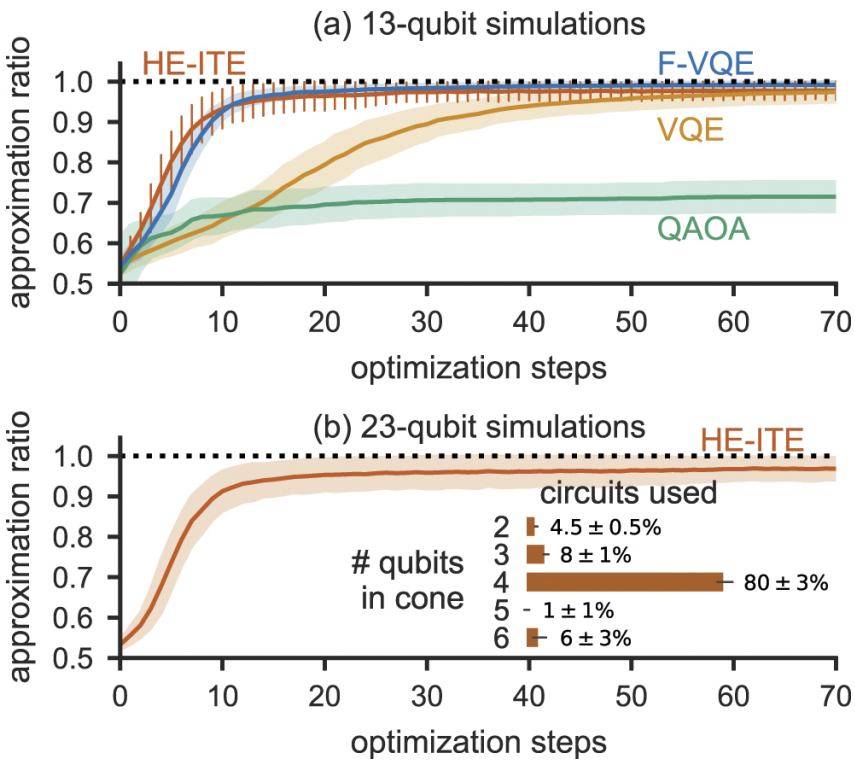
- Improvements in QAOA and VQE have largely been limited to exploring three types of graphs:
- Filtering VQE (uses an additional filtering operator to improve convergence) [8]

Current Literature

- Improvements in QAOA and VQE have largely been limited to exploring three types of graphs:
- Filtering VQE (uses an additional filtering operator to improve convergence) [8]

Current Literature

- Improvements in QAOA and VQE have largely been limited to exploring three types of graphs:
- Filtering VQE (uses an additional filtering operator to improve convergence) [8]



- Study only investigated using 3-regular graphs

Current Literature

- Warm Starting QAOA [9] – seeding initial state

Current Literature

- Warm Starting QAOA [9] – seeding initial state
  - Erdős–Rényi Graphs

Current Literature

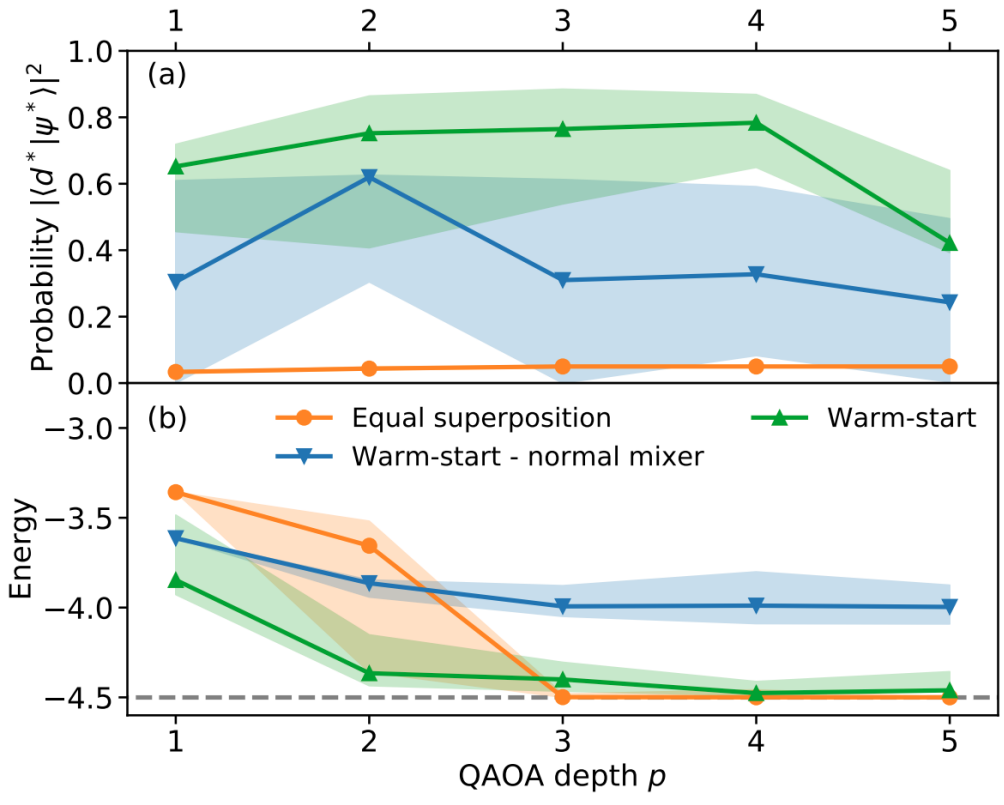
- Warm Starting QAOA [9] – seeding initial state
  - Erdős–Rényi Graphs
  - k-regular graphs

Current Literature

- Warm Starting QAOA [9] – seeding initial state
  - Erdős–Rényi Graphs
  - k-regular graphs

Current Literature

- Warm Starting QAOA [9] – seeding initial state
  - Erdős–Rényi Graphs
  - k-regular graphs



Current Literature

- Optimal QAOA parameters showing concentration over different problem instances [10], [11], [12]

Current Literature

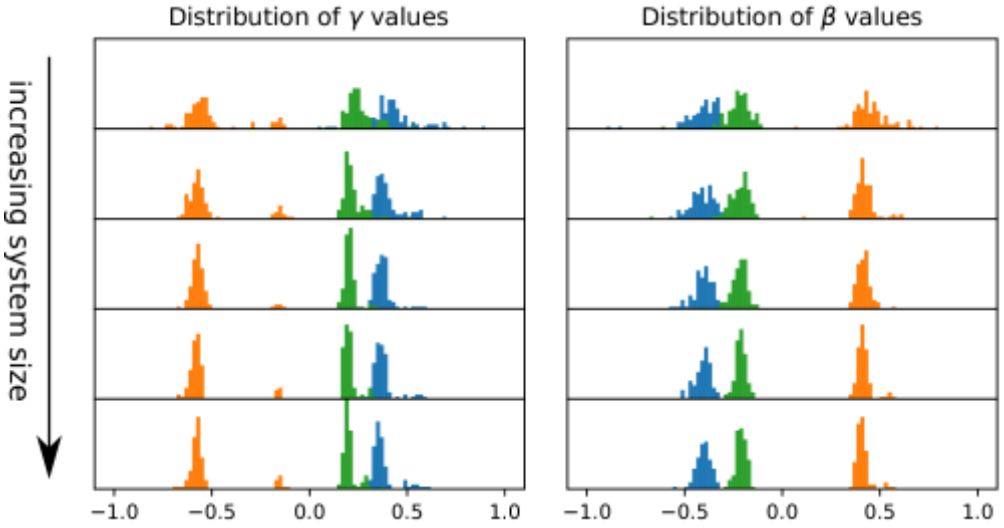
- Optimal QAOA parameters showing concentration over different problem instances [10], [11], [12]
- Study only investigated using 3-regular graphs

Current Literature

- Optimal QAOA parameters showing concentration over different problem instances [10], [11], [12]
- Study only investigated using 3-regular graphs

Current Literature

- Optimal QAOA parameters showing concentration over different problem instances [10], [11], [12]
- Study only investigated using 3-regular graphs



- **Do new developments in Quantum Algorithms in current literature extend for all classes of instances?**

Instance Space Analysis

Algorithm Selection

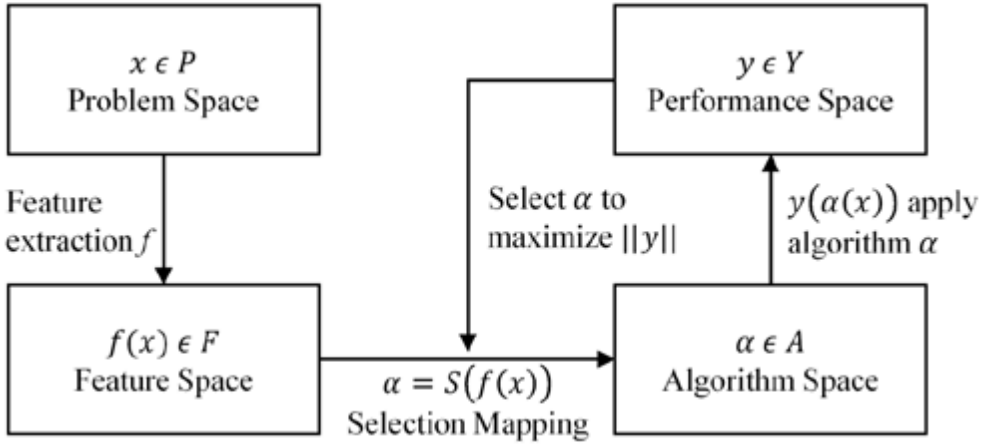
- Given a set of problem instances, predicting which algorithm is most likely to best perform was first explored by Rice [13].

Algorithm Selection

- Given a set of problem instances, predicting which algorithm is most likely to best perform was first explored by Rice [13].

Algorithm Selection

- Given a set of problem instances, predicting which algorithm is most likely to best perform was first explored by Rice [13].



- However, what we are interested in is probing the strengths and weaknesses of Quantum Algorithms for different instances of MAXCUT, TSP and 3SAT.

Instance Space Methodology

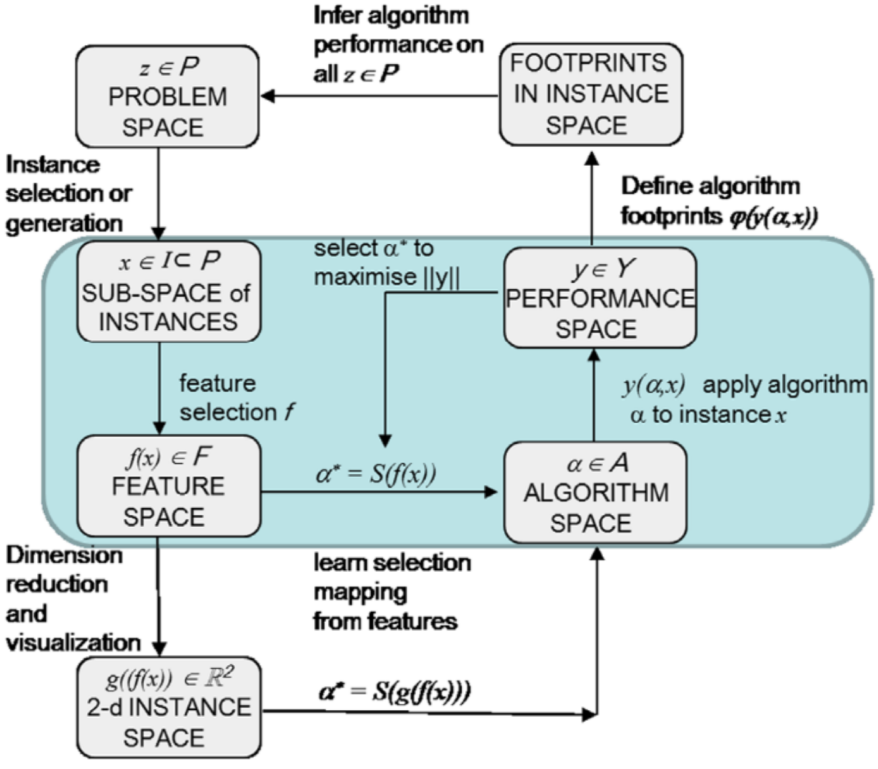
- The instance space methodology presented in [14–16] extends Rice's framework.

Instance Space Methodology

- The instance space methodology presented in [14–16] extends Rice's framework.

Instance Space Methodology

- The instance space methodology presented in [14–16] extends Rice's framework.



Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances

Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances
- The instance space $\mathcal{I} \subset \mathcal{P}$ compromises of **867** MAXCUT instances.

Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances
- The instance space $\mathcal{I} \subset \mathcal{P}$ compromises of **867** MAXCUT instances.
    - Regular Graphs

Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances
- The instance space $\mathcal{I} \subset \mathcal{P}$ compromises of **867** MAXCUT instances.
  - Regular Graphs
  - Uniform Random Graphs

Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances
- The instance space $\mathcal{I} \subset \mathcal{P}$ compromises of **867** MAXCUT instances.
  - Regular Graphs
  - Uniform Random Graphs
  - Watts-Strogatz Small World

Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances
- The instance space $\mathcal{I} \subset \mathcal{P}$ compromises of **867** MAXCUT instances.
  - Regular Graphs
  - Uniform Random Graphs
  - Watts-Strogatz Small World
  - Nearly Complete Bipartite Graphs

Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances
- The instance space $\mathcal{I} \subset \mathcal{P}$ compromises of **867** MAXCUT instances.
  - Regular Graphs
  - Uniform Random Graphs
  - Watts-Strogatz Small World
  - Nearly Complete Bipartite Graphs
  - Power Law Tree Graphs

Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances
- The instance space $\mathcal{I} \subset \mathcal{P}$ compromises of **867** MAXCUT instances.
  - Regular Graphs
  - Uniform Random Graphs
  - Watts-Strogatz Small World
  - Nearly Complete Bipartite Graphs
  - Power Law Tree Graphs
  - Nearly Complete Bipartite Graphs

Metadata: Instances

- The problem space $\mathcal{P}$ consists of all possible graph instances
- The instance space $\mathcal{I} \subset \mathcal{P}$ compromises of **867** MAXCUT instances.
  - Regular Graphs
  - Uniform Random Graphs
  - Watts-Strogatz Small World
  - Nearly Complete Bipartite Graphs
  - Power Law Tree Graphs
  - Nearly Complete Bipartite Graphs
  - Geometric Graphs

Metadata: Features

- The feature space $\mathcal{F}$ consists of 21 different instance features generated 21 features were considered

Graph Based Features	**Boolean Features**
Average Distance	Bipartite
Clique Number	Connected
Algebraic Connectivity	Acyclic
Diameter	Eulerian
Edge Connectivity	Regularity
Vertex Connectivity	Planar
Maximum Node Degree	**Laplacian (Spectral Features)**
Minimum Node Degree	Largest Eigenvalue
Cardinality of minimal dominating set	Laplacian Second Largest Eigenvalue
Number of Components	Smallest Eigenvalue
Number of vertices	
Radius	

Algorithm Portfolio

- We produce the following different instance space analyses:

Algorithm Portfolio

- We produce the following different instance space analyses:
  - QAOA, increasing layers $p = 1, ..., 10$

Algorithm Portfolio

- We produce the following different instance space analyses:
  - QAOA, increasing layers $p = 1, ..., 10$
  - QAOA, optimiser routines `COBYLA`, `NELDER_MEAD`, `SLSQP`

Algorithm Portfolio

- We produce the following different instance space analyses:
  - QAOA, increasing layers $p = 1, ..., 10$
  - QAOA, optimiser routines `COBYLA`, `NELDER_MEAD`, `SLSQP`
  - Initialisation Technique of the ground state

Algorithm Portfolio

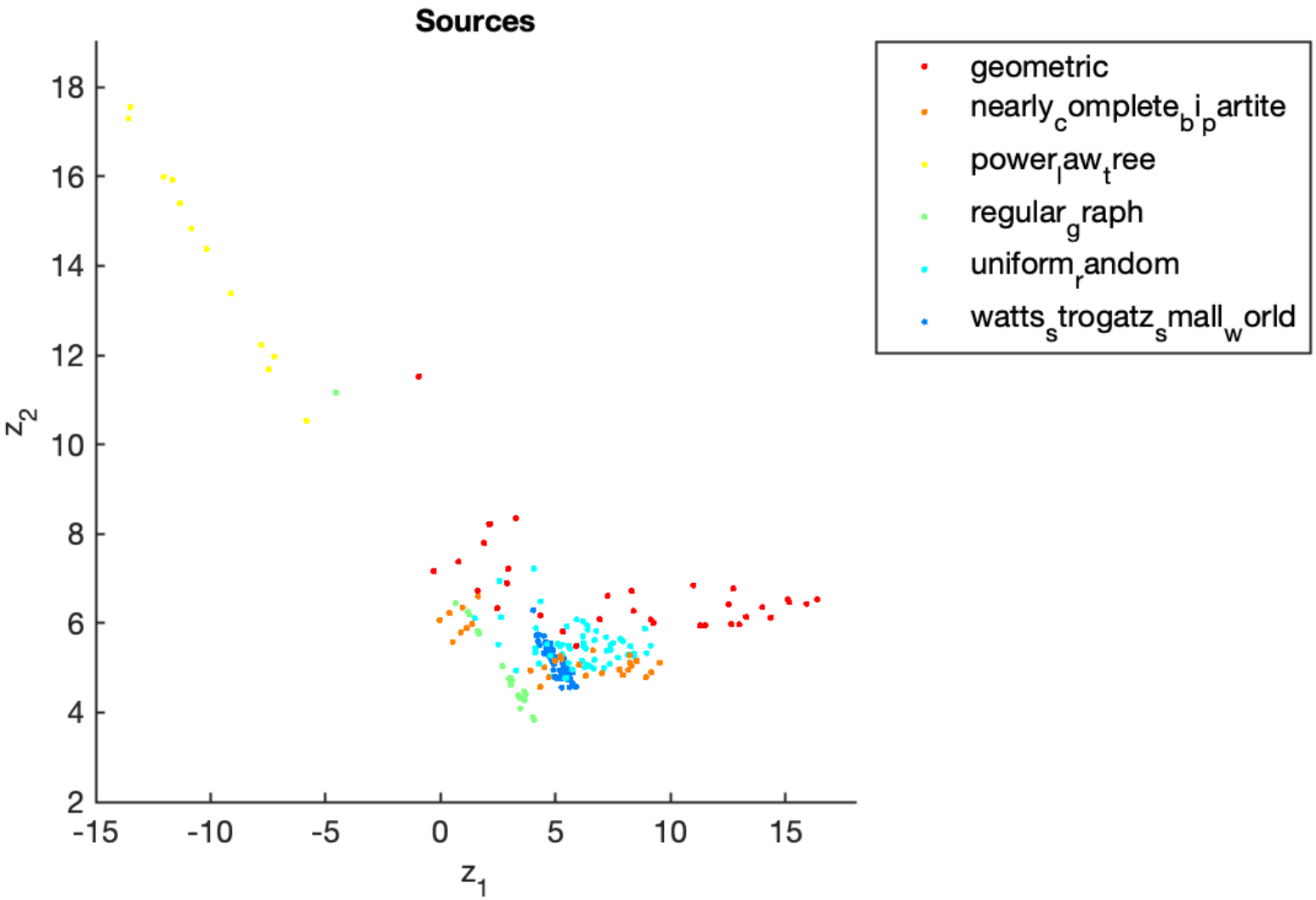
- We produce the following different instance space analyses:
  - QAOA, increasing layers $p = 1, ..., 10$
  - QAOA, optimiser routines `COBYLA`, `NELDER_MEAD`, `SLSQP`
  - Initialisation Technique of the ground state
  - Various variational techniques (VQE, F-VQE, QAOA)

Algorithm Portfolio

- We produce the following different instance space analyses:
  - QAOA, increasing layers $p = 1, ..., 10$
  - QAOA, optimiser routines `COBYLA`, `NELDER_MEAD`, `SLSQP`
  - Initialisation Technique of the ground state
  - Various variational techniques (VQE, F-VQE, QAOA)
- The performance metric $y \in \mathcal{Y}$ is energy gap ($E_{\text{gap}} = \frac{\langle\psi|H|\psi\rangle}{\langle\psi_g|H|\psi_g\rangle}$)

Results

ISA


Sources
geometric
nearly_complete_bipartite
power_law_tree
regular_graph
uniform_random
watts_strogatz_small_world
z2
z1
18
16
14
12
10
8
6
4
2
-15
-10
-5
0
5
10
15

Feature Distribution

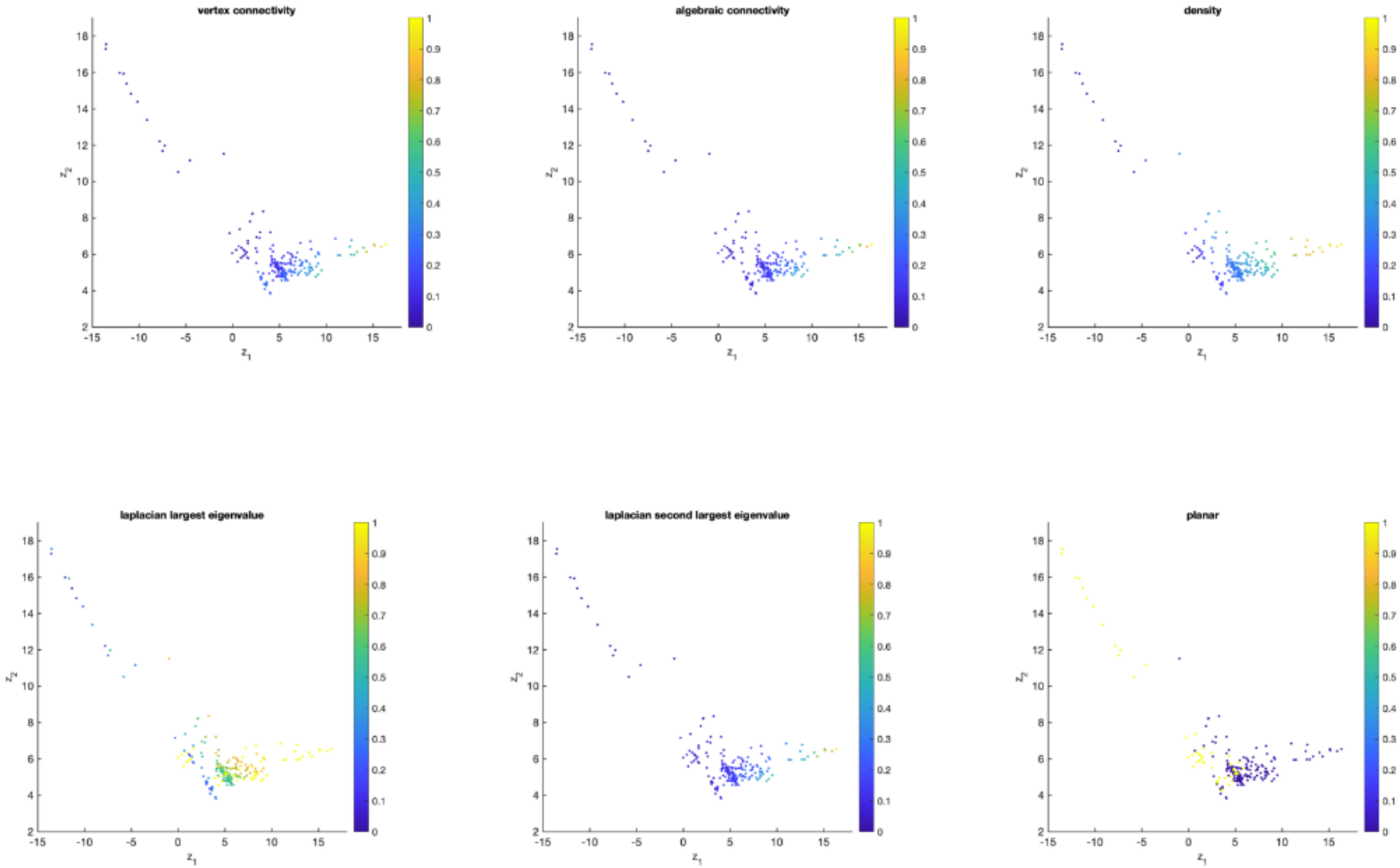
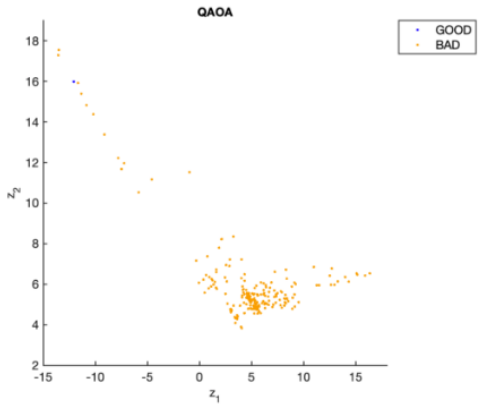
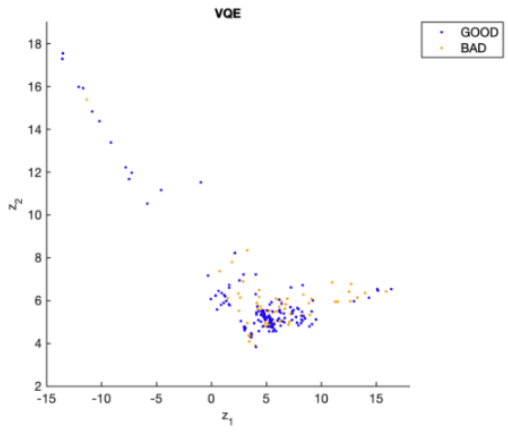
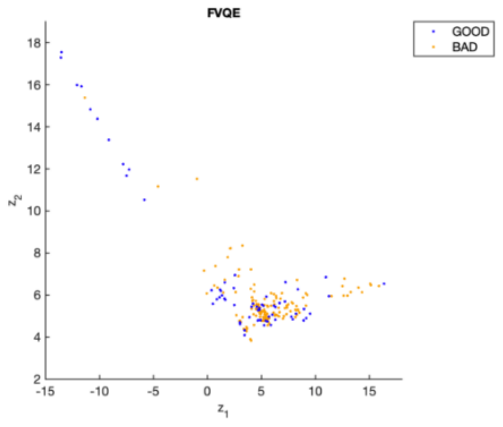


Figure 3: Features

Performance Distribution



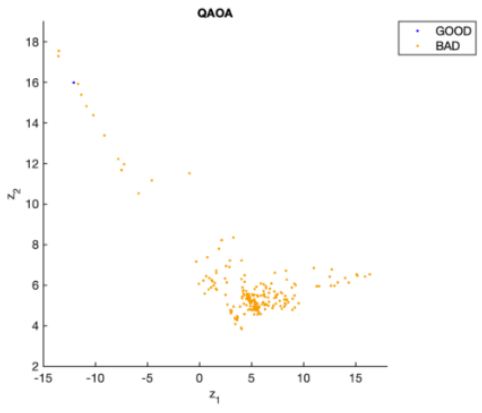


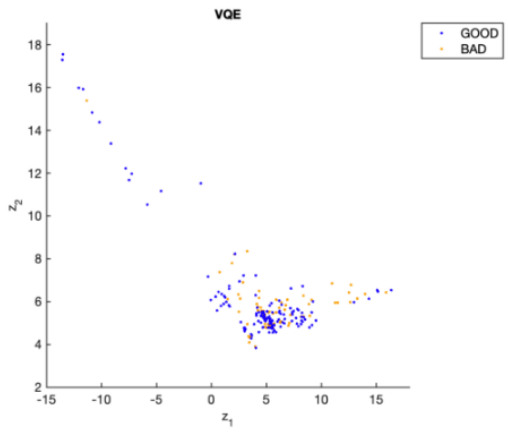


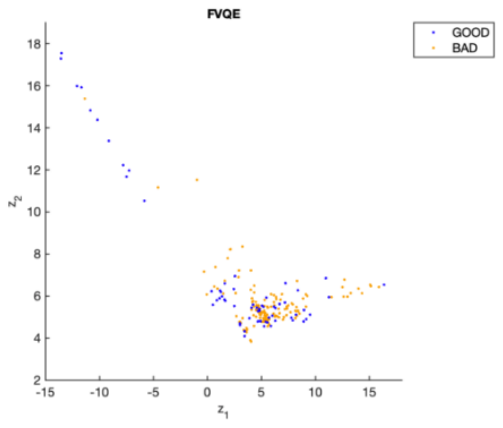
Vanilla QAOA, increasing layers

- These experiments are still currently running on SPARTAN

Performance Distribution



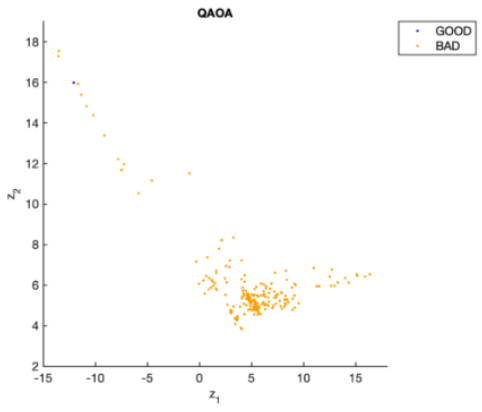


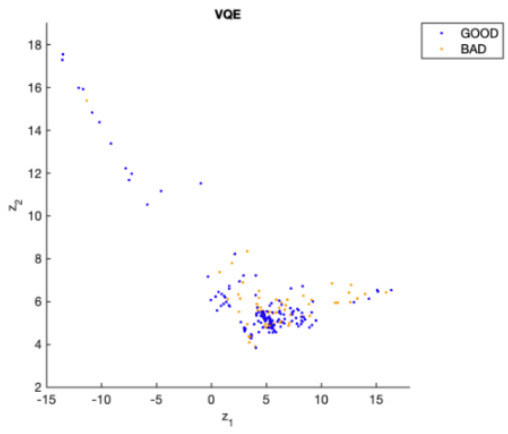


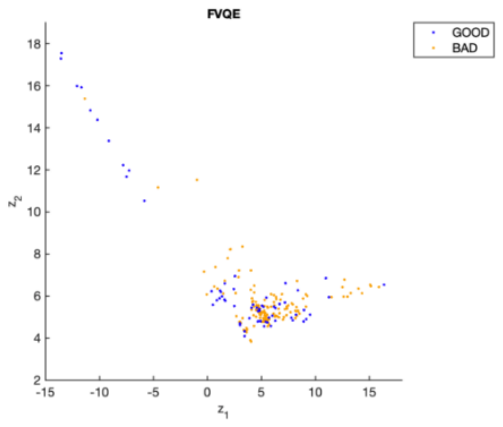
Vanilla QAOA, increasing layers

- These experiments are still currently running on SPARTAN
- Run a vanilla QAOA instance for layers $p = 1, ..., 10$

Performance Distribution



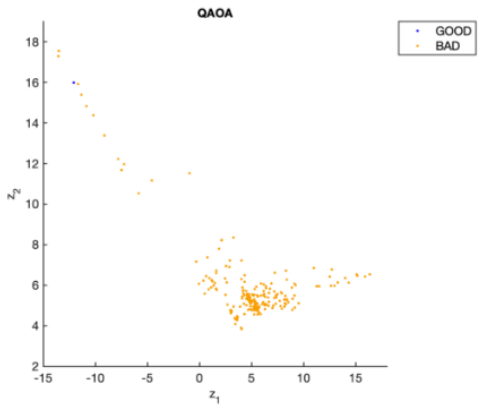


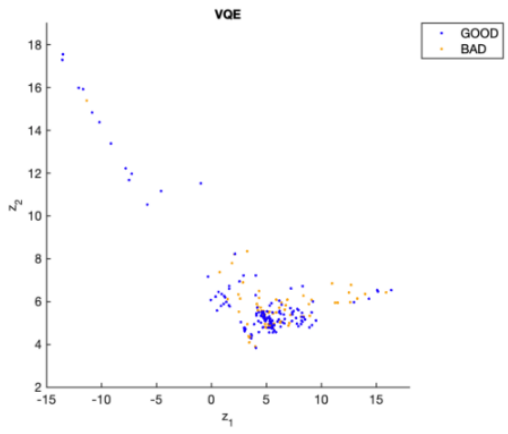


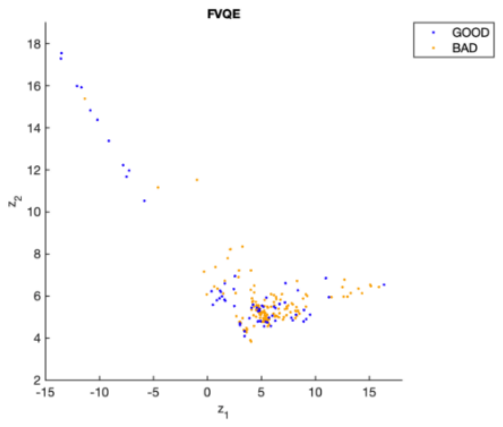
Vanilla QAOA, increasing layers

- These experiments are still currently running on SPARTAN
- Run a vanilla QAOA instance for layers $p = 1, ..., 10$
  - 5 restarts

Performance Distribution







Vanilla QAOA, increasing layers

- These experiments are still currently running on SPARTAN
- Run a vanilla QAOA instance for layers $p = 1, ..., 10$
  - 5 restarts
  - 200 13 node MAXCUT instances of each source

Vanilla QAOA with Classical Optimisation Techniques (SPSA, COBYLA, NELDER_MEAD)

- Run a vanilla QAOA instance for layers $p = 7$ (optimal result for previous)

Vanilla QAOA with Classical Optimisation Techniques (SPSA, COBYLA, NELDER_MEAD)

- Run a vanilla QAOA instance for layers $p = 7$ (optimal result for previous)
  - 5 restarts

Vanilla QAOA with Classical Optimisation Techniques (SPSA, COBYLA, NELDER_MEAD)

- Run a vanilla QAOA instance for layers $p = 7$ (optimal result for previous)
  - 5 restarts
  - 200 13 node MAXCUT instances of each source

Initialisation Techniques of the ground state (QAOA)

- Run a vanilla QAOA instance for layers $p = 7$ and a classical optimizer of `COBYLA`
- Trotterized Quantum Annealing (TQA) [17]
- Random Initialisation
- Perturb from previous layer
- Ramped up Initialisation,
- Optimise γ and β in the Fourier Space (FT)

Thesis Outline

-Thesis Structure - Introduction - Solving Optimisation Problems using a Quantum Computer - Quantum Algorithms Studied - Instance Space Analysis - ISA: MAXCUT - ISA: TSP - ISA: 3SAT - Conclusion

Discuss thesis outline

- Here is a link to the thesis outline

Next Steps

- Complete Instance Space Analysis on existing algorithms and additional adaptations (2 months)

Next Steps

- Complete Instance Space Analysis on existing algorithms and additional adaptations (2 months)
- Formalize findings into a paper (currently in progress - 1 month)

Next Steps

- Complete Instance Space Analysis on existing algorithms and additional adaptations (2 months)
- Formalize findings into a paper (currently in progress - 1 month)
- Completing and submitting thesis (3 months)

Appendix

Other Optimisation Problems Explored

Other Optimisation Problems Explored

- TSP

Other Optimisation Problems Explored

- TSP
- 3SAT

The Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP)

- The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that involves finding the shortest possible route that visits a given set of cities exactly once and returns to the starting city.

The Traveling Salesman Problem (TSP)

- The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that involves finding the shortest possible route that visits a given set of cities exactly once and returns to the starting city.
- Let $C = c_1, c_2, ..., c_n$ be a set of cities, where c_i is the i-th city.

The Traveling Salesman Problem (TSP)

- The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that involves finding the shortest possible route that visits a given set of cities exactly once and returns to the starting city.
- Let $C = c_1, c_2, ..., c_n$ be a set of cities, where c_i is the i-th city.
- Let $d(c_i, c_j)$ be a function that calculates the distance between cities c_i and c_j.

The Traveling Salesman Problem (TSP)

- The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that involves finding the shortest possible route that visits a given set of cities exactly once and returns to the starting city.
- Let $C = c_1, c_2, ..., c_n$ be a set of cities, where c_i is the i-th city.
- Let $d(c_i, c_j)$ be a function that calculates the distance between cities c_i and c_j.
- The Traveling Salesman Problem (TSP) is to find a Hamiltonian cycle $\pi = \pi_1, \pi_2, ..., \pi_n$ of minimum total length.

The Traveling Salesman Problem (TSP)

- The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that involves finding the shortest possible route that visits a given set of cities exactly once and returns to the starting city.

- Let $C = c_1, c_2, ..., c_n$ be a set of cities, where c_i is the i-th city.
- Let $d(c_i, c_j)$ be a function that calculates the distance between cities c_i and c_j.
- The Traveling Salesman Problem (TSP) is to find a Hamiltonian cycle $\pi = \pi_1, \pi_2, ..., \pi_n$ of minimum total length.
- The cycle π must visit each city in the set C exactly once and $\pi_{n+1} = \pi_1$.

The Traveling Salesman Problem (TSP)

- The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that involves finding the shortest possible route that visits a given set of cities exactly once and returns to the starting city.

- Let $C = c_1, c_2, ..., c_n$ be a set of cities, where c_i is the i-th city.
- Let $d(c_i, c_j)$ be a function that calculates the distance between cities c_i and c_j.
- The Traveling Salesman Problem (TSP) is to find a Hamiltonian cycle $\pi = \pi_1, \pi_2, ..., \pi_n$ of minimum total length.
- The cycle π must visit each city in the set C exactly once and $\pi_{n+1} = \pi_1$.
- The TSP can be defined mathematically as:

The Traveling Salesman Problem (TSP)

- The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem that involves finding the shortest possible route that visits a given set of cities exactly once and returns to the starting city.
- Let $C = c_1, c_2, ..., c_n$ be a set of cities, where c_i is the i-th city.
- Let $d(c_i, c_j)$ be a function that calculates the distance between cities c_i and c_j.
- The Traveling Salesman Problem (TSP) is to find a Hamiltonian cycle $\pi = \pi_1, \pi_2, ..., \pi_n$ of minimum total length.
- The cycle π must visit each city in the set C exactly once and $\pi_{n+1} = \pi_1$.
- The TSP can be defined mathematically as:
-

$$\min_{\pi} \sum_{i=1}^{n} d(c_{\pi_i}, c_{\pi_{i+1}})$$

TSP Example

TSP mapping to QUBO

- Represent cities as binary variables:

TSP mapping to QUBO

- Represent cities as binary variables:
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at timestep t

TSP mapping to QUBO

- Represent cities as binary variables:
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at timestep t
  - We can now represent the cost as a function of the decision variables:

TSP mapping to QUBO

- Represent cities as binary variables:
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at timestep t
  - We can now represent the cost as a function of the decision variables:
  - $$\text{Cost} = \sum_{t} \sum_{i,j \in C} d(i,j) x_{i,t} x_{j,t+1}$$

TSP mapping to QUBO

- Represent cities as binary variables:
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at timestep t
  - We can now represent the cost as a function of the decision variables:
  - $$\text{Cost} = \sum_{t} \sum_{i,j \in C} d(i,j) x_{i,t} x_{j,t+1}$$

- **Constraints:**

TSP mapping to QUBO

- Represent cities as binary variables:
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at timestep t
  - We can now represent the cost as a function of the decision variables:
  - $$\text{Cost} = \sum_{t} \sum_{i,j \in C} d(i,j) x_{i,t} x_{j,t+1}$$

- **Constraints:**
  - Each city must be visited: $\sum_{j=1}^{n} x_{ij} = 1$ for all $i \in 1, 2, \cdots, n$.

TSP mapping to QUBO

- Represent cities as binary variables:
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at timestep t
  - We can now represent the cost as a function of the decision variables:
  - $$\text{Cost} = \sum_{t} \sum_{i,j \in C} d(i,j) x_{i,t} x_{j,t+1}$$

- **Constraints:**
  - Each city must be visited: $\sum_{j=1}^{n} x_{ij} = 1$ for all $i \in 1, 2, \cdots, n$.
  - Each city must be reached from exactly one city: $\sum_{i=1}^{n} x_{ij} = 1$ for all $j \in 1, 2, \cdots, n$.

TSP mapping to QUBO

- Represent cities as binary variables:
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at timestep t
  - We can now represent the cost as a function of the decision variables:
  - $$\text{Cost} = \sum_{t} \sum_{i,j \in C} d(i,j) x_{i,t} x_{j,t+1}$$

- **Constraints:**
  - Each city must be visited: $\sum_{j=1}^{n} x_{ij} = 1$ for all $i \in 1, 2, \cdots, n$.
  - Each city must be reached from exactly one city: $\sum_{i=1}^{n} x_{ij} = 1$ for all $j \in 1, 2, \cdots, n$.
  - No two cities can be visited at the same time: $x_{ij} + x_{ji} \leq 1$ for all $i, j \in 1, 2, \cdots, n$.

TSP mapping to QUBO

- Represent cities as binary variables:
  - Assign a binary variable $x_{i,t}$ to represent which the city visited at timestep t
  - We can now represent the cost as a function of the decision variables:
  - $$\text{Cost} = \sum_{t} \sum_{i,j \in C} d(i,j) x_{i,t} x_{j,t+1}$$

- **Constraints:**
  - Each city must be visited: $\sum_{j=1}^{n} x_{ij} = 1$ for all $i \in 1, 2, \cdots, n$.
  - Each city must be reached from exactly one city: $\sum_{i=1}^{n} x_{ij} = 1$ for all $j \in 1, 2, \cdots, n$.
  - No two cities can be visited at the same time: $x_{ij} + x_{ji} \leq 1$ for all $i, j \in 1, 2, \cdots, n$.
  - To convert this problem to a QUBO express constraints as penalty terms in the objective function.

TSP mapping to QUBO

- First constraint can be expressed as $P_1 \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2$, where P_1 is a large positive constant.

TSP mapping to QUBO

- First constraint can be expressed as $P_1 \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2$, where P_1 is a large positive constant.
- Similarly, the other constraints can be expressed as penalty terms in the objective function.

TSP mapping to QUBO

- First constraint can be expressed as $P_1 \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2$, where P_1 is a large positive constant.
- Similarly, the other constraints can be expressed as penalty terms in the objective function.
- The final objective function becomes a QUBO of the form: $P_1 \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2 + P_2 \left(\sum_{i=1}^{n} x_{ij} - 1 \right)^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} P_3 (x_{ij} + x_{ji} - 1)^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$.

TSP mapping to QUBO

- First constraint can be expressed as $P_1 \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2$, where P_1 is a large positive constant.
- Similarly, the other constraints can be expressed as penalty terms in the objective function.
- The final objective function becomes a QUBO of the form: $P_1 \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2 + P_2 \left(\sum_{i=1}^{n} x_{ij} - 1 \right)^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} P_3 (x_{ij} + x_{ji} - 1)^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$.
- To simplify, we can also have $A = P_1 = P_2 = P_3$

TSP mapping to QUBO

- First constraint can be expressed as $P_1 \left(\sum_{j=1}^{n} x_{ij} - 1\right)^2$, where P_1 is a large positive constant.
- Similarly, the other constraints can be expressed as penalty terms in the objective function.
- The final objective function becomes a QUBO of the form: $P_1 \left(\sum_{j=1}^{n} x_{ij} - 1\right)^2 + P_2 \left(\sum_{i=1}^{n} x_{ij} - 1\right)^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} P_3(x_{ij} + x_{ji} - 1)^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$.
- To simplify, we can also have $A = P_1 = P_2 = P_3$
- This QUBO can then solved by a quantum algorithm

TSP mapping to QUBO

Assign a qubit to each decision variable, make a the following substitution

$$x_{l,t} = \frac{(I - Z_{l,t})}{2}.$$

Extending TSP to VRP

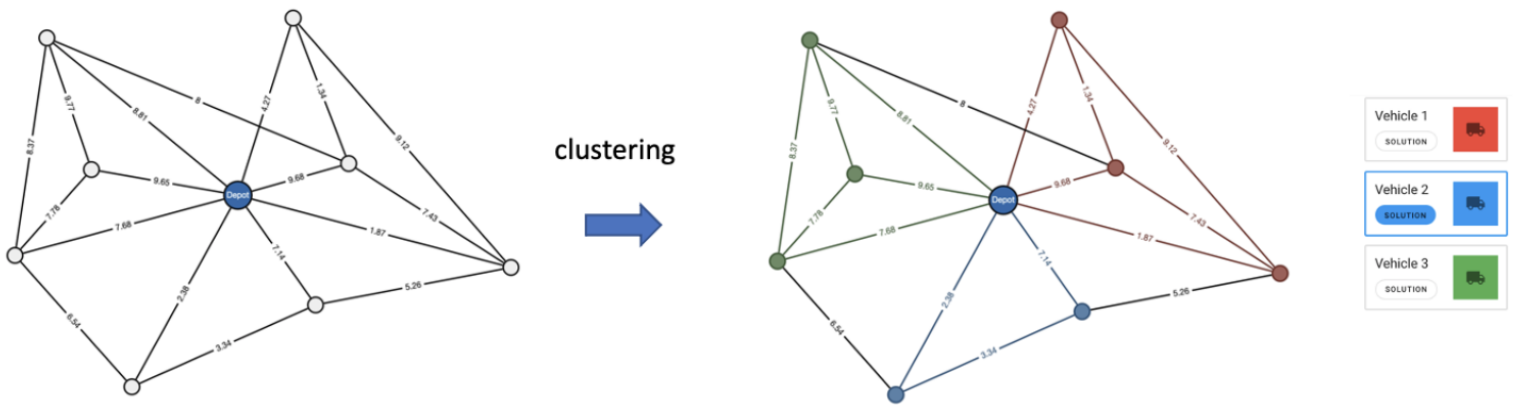
- If we have N vehicles. Implement spectral clustering create N cluster.

Extending TSP to VRP

- If we have N vehicles. Implement spectral clustering create N cluster.
- Solve each cluster independently.

Extending TSP to VRP

- If we have N vehicles. Implement spectral clustering create N cluster.
- Solve each cluster independently.
-



3SAT - Exact Cover

3SAT (Exact Cover)

- The satisfiability problem, abbreviated SAT, is a classic example of an NP-complete problem [18]

3SAT (Exact Cover)

- The satisfiability problem, abbreviated SAT, is a classic example of an NP-complete problem [18]
- The basic SAT formulation can be described as follows: Given a boolean formula (AND $\wedge$, OR $\vee$, NOT $\neg$) over n variables $(z_1, z_2, ..., z_n)$. Can one set z_i's in a manner such that the Boolean formula is true?

3SAT (Exact Cover)

- The satisfiability problem, abbreviated SAT, is a classic example of an NP-complete problem [18]
- The basic SAT formulation can be described as follows: Given a boolean formula (AND $\wedge$, OR $\vee$, NOT $\neg$) over n variables $(z_1, z_2, ..., z_n)$. Can one set z_i's in a manner such that the Boolean formula is true?
- A clause is an expression which the variables must satisfy. For example $z_1 \wedge z_2 \implies z_1 = z_2 = 1$

3SAT (Exact Cover Example)

- Consider a 4-bit number with two clauses:

3SAT (Exact Cover Example)

- Consider a 4-bit number with two clauses:
  - $z_1 \wedge z_2 \wedge z_3$

3SAT (Exact Cover Example)

- Consider a 4-bit number with two clauses:
  - $z_1 \wedge z_2 \wedge z_3$
  - $z_1 \wedge z_3 \wedge z_4$

3SAT (Exact Cover Example)

- Consider a 4-bit number with two clauses:
  - $z_1 \wedge z_2 \wedge z_3$
  - $z_1 \wedge z_3 \wedge z_4$
- Here we have 16 possible assignments, namely:

3SAT (Exact Cover Example)

- Consider a 4-bit number with two clauses:
  - $z_1 \wedge z_2 \wedge z_3$
  - $z_1 \wedge z_3 \wedge z_4$
- Here we have 16 possible assignments, namely:
-

$$\mathcal{Z} = \{\, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, \\ 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 \,\}$$

3SAT (Exact Cover Example)

- Consider a 4-bit number with two clauses:
  - $z_1 \wedge z_2 \wedge z_3$
  - $z_1 \wedge z_3 \wedge z_4$
- Here we have 16 possible assignments, namely:
-

$$\mathcal{Z} = \{\, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, \\ 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 \,\}$$

- However, our **satisfying assignment** is only $\vec{z} = 1111$

3SAT (Exact Cover Example)

- Consider a 4-bit number with two clauses:
  - $z_1 \wedge z_2 \wedge z_3$
  - $z_1 \wedge z_3 \wedge z_4$
- Here we have 16 possible assignments, namely:
-

$$\mathcal{Z} = \{\, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, \\ 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 \,\}$$

- However, our **satisfying assignment** is only $\vec{z} = 1111$
- We call a 3SAT problem with one satisfying assignment an instance of USA

3SAT (Exact Cover Example)

- Consider a 4-bit number with two clauses:
  - $z_1 \wedge z_2 \wedge z_3$
  - $z_1 \wedge z_3 \wedge z_4$
- Here we have 16 possible assignments, namely:
-

$$\mathcal{Z} = \{\, 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, \\ 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111 \,\}$$

- However, our **satisfying assignment** is only $\vec{z} = 1111$
- We call a 3SAT problem with one satisfying assignment an instance of USA
- Exact Cover implies that clauses are "exclusive" and in the form of $z_i + z_j + z_k = 1$

Mapping 3SAT to a Hamiltonian

- How do we construct H_B and H_P so that we can solve our optimisation problem?

Mapping 3SAT to a Hamiltonian

- How do we construct H_B and H_P so that we can solve our optimisation problem?
- Farhi et al.[1] describe this construction in considerable detail.

Mapping 3SAT to a Hamiltonian

- How do we construct H_B and H_P so that we can solve our optimisation problem?
- Farhi et al.[1] describe this construction in considerable detail.
- We then evolve our system with the following interpolation:

Mapping 3SAT to a Hamiltonian

- How do we construct H_B and H_P so that we can solve our optimisation problem?
- Farhi et al.[1] describe this construction in considerable detail.
- We then evolve our system with the following interpolation:
-

$$H(t) = [1 - \lambda(t)]\, H_B + \lambda(t) H_P, \qquad \lambda(t) = \frac{t}{T}$$

Mapping 3SAT to a Hamiltonian

- How do we construct H_B and H_P so that we can solve our optimisation problem?
- Farhi et al.[1] describe this construction in considerable detail.
- We then evolve our system with the following interpolation:
- $$H(t) = [1 - \lambda(t)]\, H_B + \lambda(t) H_P, \qquad \lambda(t) = \frac{t}{T}$$
- Finally, complete measurement and our solution is our final state $|\psi(t = T)\rangle$

References I

1. Farhi E, Goldstone J, Gutmann S, et al. (2001) A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem. *Science* 292: 472–475.
2. Preskill J (2018) Quantum computing in the NISQ era and beyond. *Quantum* 2: 79.
3. Raussendorf R, Briegel HJ (2001) A one-way quantum computer. *Phys Rev Lett* 86: 5188–5191.
4. Childs AM, Goldstone J (2004) Spatial search by quantum walk. *Phys Rev A* 70: 022314.
5. Karp RM (1972) Reducibility among combinatorial problems., In: Miller RE, Thatcher JW (Eds.), *Complexity of computer computations*, Plenum Press, New York, 85–103.
6. Håstad J (2001) Some optimal inapproximability results. *J ACM* 48: 798–859.

References II

7. Goemans MX, Williamson DP (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J ACM* 42: 1115–1145.
8. Amaro D, Modica C, Rosenkranz M, et al. (2022) Filtering variational quantum algorithms for combinatorial optimization. *Quantum Science and Technology* 7: 015021.
9. Egger DJ, Mareč ek J, Woerner S (2021) Warm-starting quantum optimization. *Quantum* 5: 479.
10. Farhi E, Goldstone J, Gutmann S, et al. (2022) The quantum approximate optimization algorithm and the sherrington-kirkpatrick model at infinite size. *Quantum* 6: 759.
11. Akshay V, Rabinovich D, Campos E, et al. (2021) Parameter concentration in quantum approximate optimization.

References III

12. Streif M, Leib M (2019) Training the quantum approximate optimization algorithm without access to a quantum processing unit.
13. Rice JR et al. (1976) The algorithm selection problem. *Advances in computers* 15: 5.
14. Smith-Miles K, Bowly S (2015) Generating new test instances by evolving in instance space. *Computers & Operations Research* 63: 102–113.
15. Smith-Miles K, Lopes L (2012) Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research* 39: 875–889.
16. Smith-Miles K, Baatar D, Wreford B, et al. (2014) Towards objective measures of algorithm performance across instance space. *Computers & Operations Research* 45: 12–24.

References IV

17. Sack SH, Serbyn M (2021) Quantum annealing initialization of the quantum approximate optimization algorithm. *Quantum* 5: 491.

18. Cook SA (1971) The complexity of theorem-proving procedures, *Proceedings of the third annual ACM symposium on theory of computing*, ACM, 151–158.